ESD-TR-78-98

Technical Note

1978-16

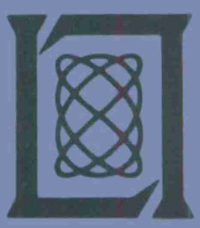
New Frontiers in Nonlinear Filtering

R. S. Bucy
K. D. Senne

26 May 1978

Prepared for the Department of the Air Force
under Electronic Systems Division Contract F19628-78-C-0002 by

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS



The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the support of the Department of the Air Force under Contract F19628-78-C-0002.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Raymond L. Loiselle, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY

NEW FRONTIERS IN NONLINEAR FILTERING

R. S. BUCY
University of Southern California

K. D. SENNE
Group 41

TECHNICAL NOTE 1978-16

26 MAY 1978

LEXINGTON MASSACHUSETTS

ABSTRACT

Examples of two and three dimensional phase demodulation problems are presented. Computer realizations for the optimal nonlinear phase estimator are discussed in detail, with emphasis on parallel computer architectures. Implementation of the nonlinear filter on various computer architectures, including the CDC6600/7600, CDC STAR-100, Illiac IV, the CRAY-1, and the Floating Point Systems AP120B is reviewed. Detailed Monte Carlo performance analysis is presented for the two-dimensional system, while partial results are included for the three dimensional case. Implications concerning the ideal computer architecture for nonlinear filter realization are discussed.

TABLE OF CONTENTS

I. INTRODUCTION

About nine years ago [6] we initiated a research effort which had as its objective the actual construction of the optimal nonlinear filter for some low state dimensional problems. The mathematical problem consisted of solving a nonlinear partial differential equation of diffusion type driven by a random process. The natural synthesis tool was a digital computer. Very quickly it was realized that the serial nature of the machines available at that time severely limited not only the realization problem but, more significantly, the error analysis problem. This is because although a highly developed theory existed, no good error performance bounds were available and error analysis could only be done by time consuming Monte Carlo simulation analysis.

We soon decided that we should concentrate our efforts on the phase demodulation problem [9] because of the exceptional amount of research effort that had been devoted to threshold extension of the classical phase lock loop, which in actuality is a suboptimal filter for the phase demodulation problem [3]. Further threshold extension had payoffs which would justify the design of a special purpose black box to realize the optimal demodulator. Also it became obvious that some thought should be given to the architecture of the black box in order that it provide real-time realization as well as effective off-line error analysis [34] capabilities. We have studied the architecture question by gaining experience with designing fast software for various parallel, pipeline and array processors, and along the way documenting the achievable error variance improvement possible by the use of the optimal demodulator.

Our purpose in undertaking this research effort is to develop a new technique for system design based on parallel computation and to show in the case of phase demodulation how the nonlinear filter can improve system performance. To those who believe our results are impractical because of the number of megaflops necessary to compute the relevant conditional density, we feel it suffices to note the enormous progress in computer speed and design in the last nine years, in order to underscore the declining validity of that argument as a function of time.

II. PROBLEM DESCRIPTION

We model the observation process as the solution of random differential equations in the sense of Ito [1]:

$$\begin{aligned} dz^1 &= H(x_3)\cos(x_1)\,dt + d\nu^1 \\ dz^2 &= H(x_3)\sin(x_1)\,dt + d\nu^2 \end{aligned} \tag{2.1}$$

where x_1 is a phase process, x_3 is the amplitude process, and ν^1 and ν^2 are Brownian motions of spectral density r which are independent of the amplitude and phase signal processes. Our object is to find optimal estimates of the current value of phase and amplitude based on past and present observations of z^1 and z^2. In order to proceed we must model the amplitude and phase processes, which may also be done in the sense of Ito as

$$\begin{aligned} dx_1 &= x_2\,dt \\ dx_2 &= dw_2 \\ dx_3 &= -\beta x_3\,dt + dw_3 \end{aligned} \tag{2.2}$$

where dw_3 and dw_2 are white noises with spectral parameters q_{33} and q_{22}, respectively. We consider two special cases of (2.1), (2.2): for the two-dimensional process we replace the $H(x_3)$ with unity and eliminate the equation for x_3 from the state equations; for the three-dimensional process we implement (2.1) and (2.2) as shown with $H(x_3) = \exp(x_3)$.

The solution of the filtering problem involves the determination of the conditional probability density for the signal process given the present and past observations. This solution is obtained by solving the following differential equation:

$$dp = Apdt + (\underline{h}-\hat{\underline{h}})' R^{-1} (d\underline{z} - \hat{\underline{h}}dt)p \tag{2.3}$$

with $\quad p: R^n \times R^+ \to R^+$ and $\underline{h}: R^n \to R^s$

where s=2 and n=3 for the three-dimensional model or n=2 for the two-dimensional model,

$$A = -\beta x_3 \frac{\partial}{\partial x_3} + x_2 \frac{\partial}{\partial x_1} + \frac{1}{2} q_{33} \frac{\partial^2}{\partial x_3^2} + \frac{1}{2} q_{22} \frac{\partial^2}{\partial x_2^2}$$

$$\underline{h} = \begin{pmatrix} H(x_3) \cos (x_1) \\ H(x_3) \sin (x_1) \end{pmatrix}, \quad R = \begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix} \quad , \tag{2.4}$$

$$\underline{z} = \begin{pmatrix} z^1 \\ z^2 \end{pmatrix}$$

Equation (2.3) is the Stratonovich-Kushner equation for the solution to the nonlinear filtering problem (see [4]).

Two alternatives exist for solving (2.3) on a digital computer [2]. One scheme involves the direct replacement of (2.3) with a suitable finite difference equation which in the continuous limit approaches (2.3), and when solved yields a solution which also in the limit (hopefully) approaches the solution of (2.3). A more effective method, however, is to pose and solve a sequence of discrete filtering problems as

$$\begin{aligned} z_n^1 &= H(x_n^3) \cos (x_n^1) + \nu_n^1 \\ z_n^2 &= H(x_n^3) \sin (x_n^1) + \nu_n^2 \end{aligned} \tag{2.5}$$

$$\begin{aligned} x_n^1 &= x_{n-1}^1 + \Delta x_{n-1}^2 \\ x_n^2 &= x_{n-1}^2 + W_{n-1}^2 \\ x_n^3 &= x_{n-1}^3 + \beta \Delta x_{n-1}^3 + w_{n-1}^3 \end{aligned} \tag{2.6}$$

where $E(\nu_n^i)^2 = r/\Delta$, $E(w_n^i)^2 = \Delta q_{ii}$ with Δ interpreted as the sampling interval. The solution to the discrete filtering problem evolves according to the equations [4], [8]

$$P_{n+1} = S * F_n \tag{2.7}$$

$$F_n = \frac{1}{K_n} D_n \cdot P_n = \frac{1}{K_n} D_n \cdot S * F_{n-1} \tag{2.8}$$

where P_n $\{F_n\}$ is the conditional distribution of x_n^1, x_n^2, and x_n^3 given z_{n-1}^i, z_{n-2}^i, ..., z_o^i $\{z_n^i, z_{n-1}^i, \ldots, z_o^i\}$, * denotes convolution, $\cdot$ denotes pointwise multiplication. The functions S and D_n are derived from (2.5) and (2.6), based on assumed probability density functions (Gaussian) for w_n^i and ν_n^i, respectively, and K_n is a scalar which is chosen to normalize F_n to have unit total mass.

For the two-dimensional special case (i.e., without the amplitude signal process) the optimal filter recursion of (2.8) becomes

$$F_n^*(y_1,y_2) = \frac{1}{K_n} D_n(y_1) \int_{-\infty}^{\infty} \exp\left\{\frac{(y_2-\mu)^2}{2q_{22}\Delta}\right\} F_{n-1}^*(y_1-\mu\Delta,\mu)d\mu \tag{2.9}$$

where

$$D_n(y_1) = \exp\left\{\frac{z_n^1 \cos(y_1) + z_n^2 \sin(y_1)}{r/\Delta}\right\} \tag{2.10}$$

It can be shown [32], [35] that a cyclically modulated density F_n, defined as

$$F_n(y_1,y_2) \triangleq \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} F_n^*(y_1 + 2\pi k, y_2 + 2\pi\ell/\Delta) \tag{2.11}$$

with $-\pi \leq y_1 < \pi$, and $-\pi/\Delta \leq y_2 < \pi/\Delta$, carries all of the information necessary for nonlinear filtering subject to the cyclic loss function $L(\varepsilon_1) = \frac{1}{2}(1-\cos(\varepsilon_1))$,

where ε_1 is the error in the estimate of the phase x_1. The modulated density satisfies the recursion relation

$$F_n(\sigma, \tau) = \frac{1}{K_n} D_n(\sigma) \int_{-\pi/\Delta}^{\pi/\Delta} S(\tau-\mu)F(\sigma-\mu\Delta,\mu)d\mu \tag{2.12}$$

where

$$S(\tau-\mu) = \sum_{i=-\infty}^{\infty} \exp\left\{ - \frac{(\tau-\mu + 2\pi i/\Delta)^2}{2q_{22}\Delta} \right\} . \tag{2.13}$$

It has also been established [32], [35] that the estimate x_n^* of x_n^1 which minimizes the cyclic loss function is given by

$$x_n^* = \arg\left\{ E\, e^{ix_n^1} \,|z_i^1 , z_i^2 \quad i \leq n) \right\} . \tag{2.14}$$

The optimal filter recursion (2.8) for the three-dimensional problem can be obtained in an analogous fashion. Using a discrete form of the representation theorem (see [4]) and following the development of the two dimensional special case, the density F_n for the filtered amplitude and cyclic phase processes can be updated recursively as

$$F_n(\sigma,\tau,\alpha) = \frac{1}{K_n} D_n(\sigma,\alpha) \int_{-\infty}^{\infty} \int_{-\pi/\Delta}^{\pi/\Delta} S_1(\tau-\alpha) S_2(\alpha-\beta\eta) \cdot$$

$$F_{n-1}(\sigma-\mu\Delta, \mu, \eta) d\mu d\eta \qquad (2.15)$$

where

$$D_n(\sigma,\alpha) = \exp\left[\frac{\Delta}{r} H(\alpha) \left(z_n^1 \cos\sigma + z_n^2 \sin\sigma - \frac{1}{2} H(\alpha)\right)\right]$$

$$S_1(\tau-\alpha) = \sum_{k=-\infty}^{\infty} \exp\left[-\frac{1}{2q_{22}\Delta} \left(\tau-\alpha+\frac{2\pi k}{\Delta}\right)^2\right]$$

$$S_2(\alpha-\beta\eta) = \exp\left[-\frac{1}{2q_{33}\Delta} (\alpha-\beta\eta)^2\right]$$

and K_n is again a normalizing constant chosen so as to make F_n integrate to unity.

As discussed previously, the most convenient and accurate scheme for solving the continuous-time filtering problem is to solve the corresponding discrete-time filtering problem for appropriately small Δ, instead of attacking the continuous-time problem directly. Indeed, it is interesting to observe the relationship between (2.3) and the pair (2.7) and (2.8). Solving (2.7) and (2.8) n times leads to

$$P_n = (e^{A\Delta}_1 e^{B\Delta}_1)\ (e^{A\Delta}_2 e^{B\Delta}_2)\ \ldots\ (e^{A\Delta}_n e^{B\Delta}_n)\ P_o \tag{2.16}$$

where the operator $e^{A\Delta}$ is convolution with S and $e^{B\Delta}$ is multiplication by D_n. The infinitesimal generators of the operators e^{At} and e^{Bt} are A (see 2.4) and B: $p \rightarrow (\underline{h}-\hat{\underline{h}})' R^{-1} (d\underline{z}-\hat{\underline{h}}dt)p$, corresponding to the terms on the right side of (2.3). The fact that the relation for P_n approaches the solution P(t,x) of (2.3) is known as Trotter's formula. Since we replace the continuous-time problem with the analogous discrete-time problem, the update relations (2.7) and (2.8) inherit the properties of P(t,x); in particular, the positivity of P(t,x) implies the same for P_n. It is of course important to preserve this positivity relationship when additional approximations are introduced for the purpose of realizing (2.7) and (2.8) on a digital computer.

We have considered various ways to represent the conditional density in the update formulae (2.7) and (2.8) by a finite number of finite precision parameters. The representations will be briefly reviewed in the context of the two-dimensional phase demodulation problem. In the point-mass representation (see [18]) the conditional density is characterized by a finite set of weights placed at discrete points in the domain of the density. A fixed rectangular grid, consisting of m points in the phase variable x_1 and n points in the phase rate variable x_2, is defined as follows, with indices (i,j) corresponding to the state variables (σ,τ), respectively:

$$\sigma(i) = -\pi + 2\pi\left(\frac{i}{m}\right) + \pi\left(\frac{1}{m}\right) = \pi\left(\frac{1+2i}{m} - 1\right), \tag{2.17}$$

$$i=0,\ldots,m-1$$

$$\tau(j) = \frac{-\pi}{\Delta} + \frac{2\pi}{\Delta}\left(\frac{j}{n}\right) + \frac{\pi}{\Delta}\left(\frac{1}{n}\right) = \frac{\pi}{\Delta}\left(\frac{1+2j}{n} - 1\right), \tag{2.18}$$

$$j=0,\ldots,n-1$$

Note that the phase rate variables, evaluated at the points $\tau(i)$ may be used directly in the convolution (2.12), but the a priori conditional density F_{n-1} must be interpolated along the phase coordinate such that $\xi(k) = \sigma(i) - \Delta\tau(j)$, so that k must correspond to pairs i and j which satisfy

$$k = i + \frac{m}{2} - \frac{m}{n}\left(\frac{1}{2} + j\right) \tag{2.19}$$

Now if m is arbitrarily taken to be an even integer and n/m is taken to be an integer then (2.21) may be decomposed into an integer part and a fractional part as follows.

$$k = \underline{k} + \Delta k, \tag{2.20}$$

where

$$\underline{k} = \left[i + \frac{m}{2} - 1 - j \text{ DIV}\left(\frac{n}{m}\right)\right] \text{ MOD } m \tag{2.21}$$

$$\Delta k = \left[\frac{1}{2} + j \text{ MOD}\left(\frac{n}{m}\right)\right]/\frac{n}{m} \tag{2.22}$$

$i=0, \ldots, m-1, \; j=0, \ldots, n-1$

DIV is integer division (with discarded remainder)
MOD is the remainder after integer division

The interpolated density point will lie between $\underline{k}$ and $(\underline{k}+1)$MOD m in the phase coordinate, with weighting Δk on the former and $1-\Delta k$ on the latter. It can be seen from examining (2.19)-(2.23) that interpolation in the phase coordinate is always required if m is even and n>m. Moreover, if m is odd, interpolation will also be required unless m=n, in which case

$$k = \left(i-j + \frac{m-1}{2}\right) \text{ MOD } m \tag{2.23}$$

Previous simulation results [31], [32], [35] have been reported using m odd with n/m chosen between 4 and 6. The present work takes m to be even (32) in order to provide the best mapping to machines such as Illiac IV which have an even number of processing elements. There is no other significance to this change of realization. In any event, with n/m=4 there must be interpolation independent of whether m is odd or even.

The term (2.13) consists of an infinite sum. The values which the argument may assume consist only of integer multiples of $2\pi/n\Delta$ in the discrete coordinate space, resulting in

$$S(p) = \sum_{\ell=-\infty}^{\infty} \exp\left[-\frac{2}{q_{22}\Delta}\left(\frac{\pi}{\Delta}\right)^2\left(\frac{p}{n}+\ell\right)^2\right] \qquad (2.24)$$

$$|p| = 0,1, \ldots, n-1$$

The expression (2.24) reveals that S is an even function which is cyclic modulo n. Further, for small values of q_{22}, the contribution to the sum (2.24) of all terms except $\ell=0$ is neglible. A key result for the two-dimensional problem [41] is that the nonlinear filter performance depends only on a function of q_{22} and r; thus, any problem may be scaled by adjusting both r and q_{22} so as to provide a suitably small q_{22} so as to guarantee the validity of the approximation

$$S(p) \cong \exp\left[\frac{-2}{q_{22}\Delta}\left(\frac{\pi}{\Delta}\right)^2\left(\frac{p}{n}\right)^2\right], \quad |p| = 0,1, \ldots, n-1 \qquad (2.25)$$

In the examples chosen for this paper, q_{22} is taken to be 0.01, which permits us to ignore all terms for $|p| > 5$, that is $S(p) \overset{\Delta}{=} 0$ for $|p| > 5$.

The point mass method will be the representation discussed in detail in this paper because it is accurate, preserves positivity, readily generalizes to higher dimensions and is easily implemented on a large class of digital computers (see [23], [27], [28], and [29]). We will briefly highlight in the

remainder of this section some other representations which have previously been studied in connection with this problem (see [5], [7], [11], [13], [14], [16], [20], [21], [25], [26], [30], [32], [35], [36], [37], [39], [40], and [44]).

The doubly periodic density function $F_n(x,y)$ may be represented by a finite number of Fourier coefficients $\{a_{ij}(n)\}_{(i,j)\in S}$, where S is finite. Moreover, it is possible to derive a recursive update so that $\{a_{ij}(n+1)\}$ can be obtained directly from $\{a_{ij}(n)\}$, and the cyclic optimal estimate can be expressed as $x_n^* = \arg [a_{-1,0}(n)]$. For more details, consult [32], [35], [36], and [44]. Experience has shown that the behavior of the Fourier filter is very good with respect to the estimates, in spite of considerable negativity introduced by the truncated series representation. Moreover, for reasonably low signal-to-noise conditions the filter can be as much as an order of magnitude faster than the equivalent point-mass filter. The speed advantage begins to disappear, however, as the signal/noise environment is improved, and the filter requires the use of complex arithmetic and Bessel functions, which makes it somewhat more difficult to realize, and unattractive to vectorize.

If F_n is represented by a linear combination of a finite number of splines under tension, the update formula (2.12) can be used to obtain a corresponding recursive updating formula for the spline coefficients. The spline filter can be made to operate as fast as the Fourier filter and the positivity of the densities can be assured by appropriate (experimental) adjustment of the tension coefficients. The implementation of the spline filter, however, requires considerable nonvectorizable overhead so that implementation of filters on highly parallel or vectored computers would not provide significant speed improvement.

Another class of representations involves fitting densities with Gauss-Hermite polynomials or by sums of Gaussian densities. These methods show the greatest promise when the signal/noise environment is rather good. The choice of the appropriate number of terms and the placement of the terms (in the case of Gaussian sums) requires a good deal of experimentation and

nonvectorizable computation as the signal to noise ratio drops well below the threshold of the phase-locked loop.

After much study, therefore, the point-mass filter has evolved to a standard of comparison for other techniques. Moreover, as computer evolution continues a higher premium is being placed upon regularity and parallelizability of computation, so that the comparative importance of the point-mass method will continue to grow (see also [17]). Since the point mass filter also enjoys uniform convergence (as the density of the grid increases) and overall simplicity of implementation it is natural to do a computer architecture tradeoff in the context of the point-mass filter.

III. CANDIDATE COMPUTER ARCHITECTURES

As discussed in the previous section, it is possible to increase the computational efficiency of some examples of our problem by up to an order of magnitude by the use of sophisticated numerical techniques. The standard for performance comparison, however, remains the point-mass filter which, coincidentally, also has a simple and highly regular realization. Since performance analysis for nonlinear filters requires extensive Monte Carlo simulations, and the point-mass filter is a reference standard, the point-mass filter must be efficiently implemented. Thus, our attention has concentrated on the subject of computer architectures which can exploit the regularity and parallelizability of the point-mass filter. We are of course interested both in the suitability of the architecture and the ease of programmability for our class or problems.

The majority of commercially available computers are essentially single thread or Single-Instruction-Single-Data (SISD) architectures. Nevertheless, some reasonably fast machines of this type have been built. By employing ever faster memories, high-speed cache memories, instruction fetch-decode-execute overlap, fast multipliers, etc. various serial machines have become quite competitive for this problem, at least for the two-dimensional version. Examples of such machines are the IBM 370/168 and the PDP 11-70, both of which have been used in this study. On the other hand, extension of the study of nonlinear filter performance to problems of higher dimension clearly requires more powerful architectures (see also [22], [24], [38], [43]).

The vector processing machines which have been studied for this report include two broad categories: the array processor and the linear vector pipeline processor. An array processor such as the Illiac IV makes use of many identical processors to create a Single-Instruction-Multiple-Data (SIMD) environment. By contrast, vector pipeline machines such as the CDC STAR and TI ASC make use of memory paging and segmented arithmetic functional units to increase the rate of throughput of multiple identical computations on the corresponding elements of vector operands. The vectors for pipeline processors may be obtained from sequential memory locations or from linearly related memory locations, in

general, forming what has come to be referred to as a linear vector. The vectors might also be taken from arbitrary memory locations by vector indexing, but any potential advantage of memory paging might thereby be lost.

As experience with advanced architectures accumulates (see [42]), it has become apparent that a multiplicity of architectural features must be simultaneously present in order to create a truly general purpose environment. The earliest examples of this trend were the look ahead machines such as the CDC 6600/7600 which incorporated a finite instruction stack with multiple functional units scheduled by means of an automatic reservation system. More recently, Floating Point Systems has introduced the AP120B which employs user-generated horizontal microcode to permit the simultaneous execution of a number of parallel activities, including a floating point add, a floating multiply and several register-register and register-memory transfers.

The latest and (to date) most impressive hybrid machine architecture to be developed is the CRAY 1, which combines the look-ahead reservation concept with a large variety of registers, vector registers, and multiple segmented functional units to achieve a very general purpose machine. The segmentation of all of the functional units permits pipelining from vector registers which in turn can be filled with linear vectors from memory. Functional unit reservations permit chaining of nonconflicting sequential operations, thereby combining overhead from several pipeline operations into one chained operation. Another hybrid architecture involving the array concept is being developed by Burroughs (the BSP [54]).

In the remainder of this section we review the computational requirements of the point-mass filter and summarize the impact of the various computer architectures on the realization of this filter.

A. Assessment of Required Computations

The various manipulations of the pointmass densities which are necessary in order to perform the convolution by S are depicted in Figures 1 and 2. Before the convolution can be done it is necessary to develop an interpolated F_{n-1} density as follows: imagine the cyclic density attached to a elastic cylinder with axis aligned along the phase-rate coordinate (see Fig. 1). The coordinate transformation leading to the interpolated F_{n-1} is imagined as follows: fasten the ends of the elastic cylinder to parallel flat plates; now rotate the plates in opposite directions each one-half revolution; next compute the interpolated density along lines parallel to the cylinder axis. The interpolated density on the resulting cylinder is fitted with a gaussian sleeve (i.e., height equal to amplitude of S), and the cylinder is joined end-to-end, forming a torus. The convolution operation produces a new density function by replacing the ring under the center of gaussian sleeve with the integral of the product of the sleeve height with the original density under the sleeve (see Fig. 2). The final operations in the implementation of the cyclic nonlinear filter are the multiplication by D_n and the normalization by K_n. The estimates are computed by collapsing the resulting density along the phase-rate dimension, multiplying the resulting ring by sine and cosine of phase and integrating, and finally by computing the arctangent of the ratio of the results.

The computations required to implement the two-dimensional point mass filter are summarized in Table 1, as a function of m and n. The sensor terms are the only ones which require math functions (exponentials). Since only m exponentials are required, this computation is generally insignificant compared with the overall filter update, so no special effort has been expended to optimize the required computations.

B. The Illiac IV Algorithm

The primary considerations for programming the Illiac IV array are the proper utilization of all of the 64 Processor Elements (PEs) and minimization of data routing between PEs (see [33], [46], [50]). In order to accomplish the efficient use of the PEs, the values of m=32 and n=128 were selected and

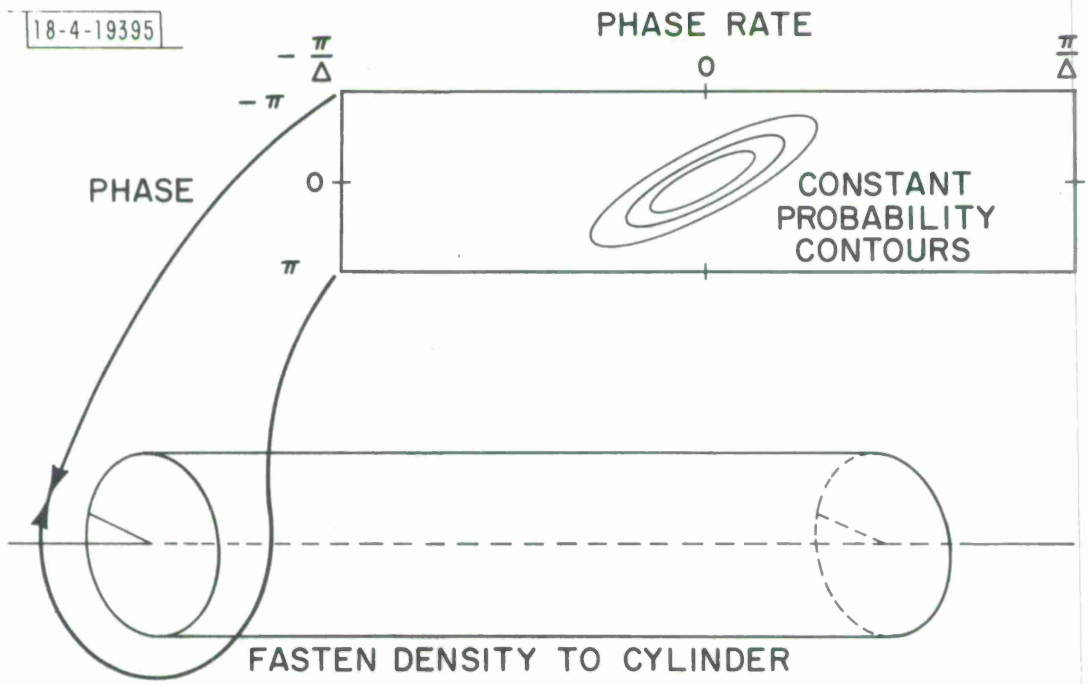


Fig. 1. First step in convolution process.

18-4-19396

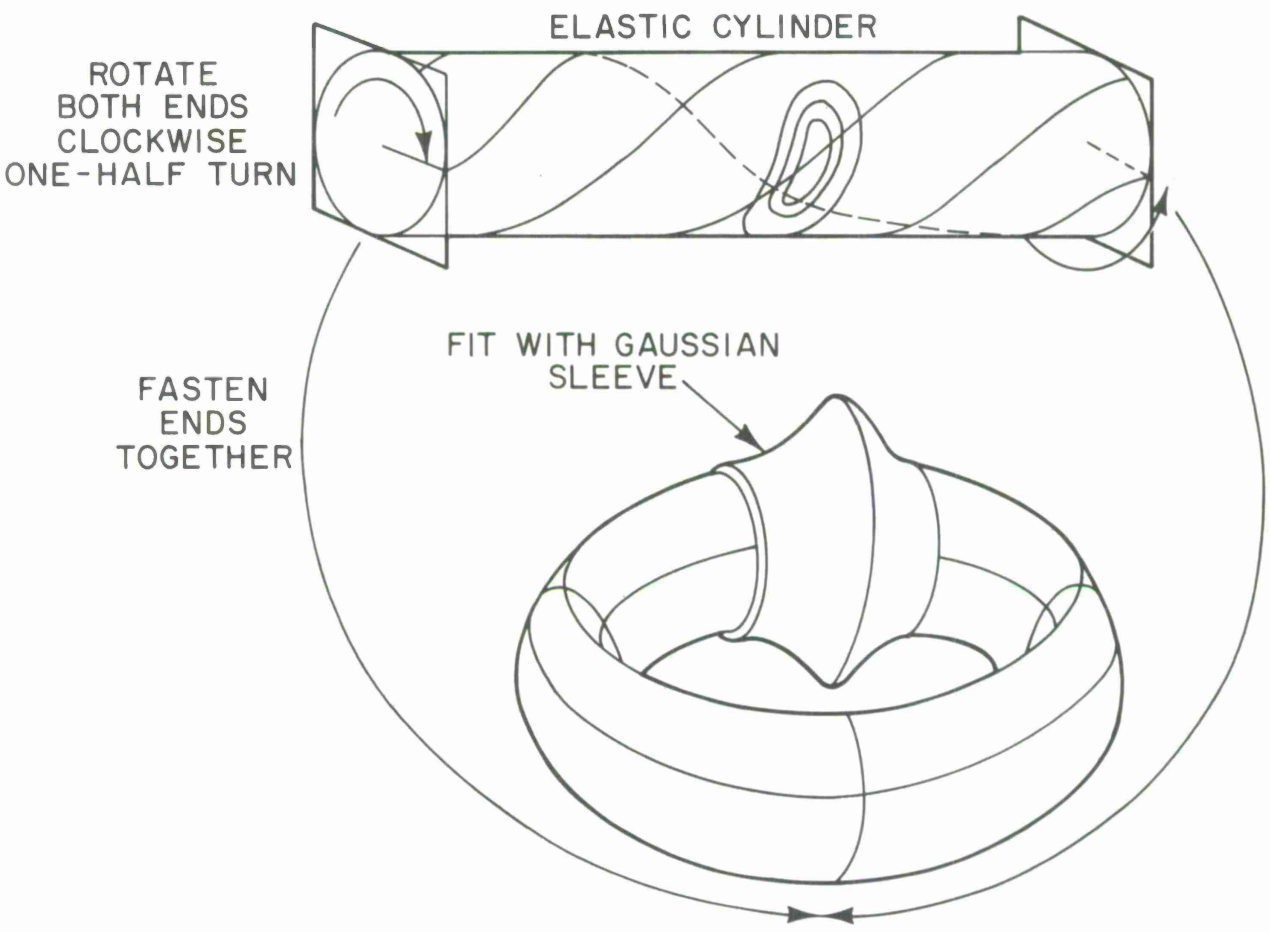


Fig. 2. Phase rotation and phase-rate convolution.

TABLE 1

FLOATING-POINT OPERATIONS FOR FILTER

Function	Number of Operations			
	Multiples	Adds	Divisions	Exponentials
Sensor Terms	2m	m	0	m
Interpolation	mn	2mn	0	0
Convolution	5mn	10mn	0	0
Row Sums	0	mn-m	0	0
Estimates	3m	3m-3	1	0
Normalization	mn+m	0	0	0
Total	7mn+6m	13mn+3m-3	1	m
Example	28864	53341	1	32
m = 32 n = 128	82206			

utilized for all machine examples. On the Illiac, two rows of PE storage were used for each value of the phase samples in F_n. The interpolation and convolution were accomplished in a totally parallel fashion, using all 64 PEs, except for the operation depicted in Fig. 3. It is necessary to perform a cyclic rotation of each phase row to accumulate the terms for the convolution (a cyclic rotation to the right is combined with a cyclic rotation to the left at each step). Since a cyclic routing on Illiac IV involves only one row at a time, it was necessary to form the two-row rotation by two single-row rotations, followed by an end-element switch (involving three transfers with only one PE enabled).

The overall effectiveness of the Illiac IV algorithm is evident from the fact that so few operations are required which involve fewer than 64 PEs enabled. The sensor terms are computed with only 32 PEs enabled, but this operation only involves about 5% of the estimate update. The single PE transfers and the convolution are also only responsible for about 5% of the computation time. Finally, the row sums are done logarithmically with a PE utilization efficiency of 16.7%, and they account for another 5% of the estimated computation time. Thus, the net PE utilization efficiency of this algorithm is 87.5%.

Although the Illiac IV algorithm is capable of utilizing all of the PEs 85% of the time, it remains to estimate what percentage of the PE time is devoted to floating point computation. The Illiac architecture permits only one floating point operation to be in progress at one time in each PE. At least one operand fetch from memory can be overlapped with computation, however, so that throughput can be maximized by chaining multiple arithmetic operations in series and retaining intermediate results within PE registers. Assuming two operands must be fetched at 10 cycles each to start a chain and that one result must be stored for 10 cycles to end the chain, then the megaflop rate (i.e., millions of floating point operations per second) for a chain of N identical 10-cycle operations (e.g., floating point multiplies) is given by

$$R = \frac{6400}{C} \left(\frac{N}{N+3}\right) \qquad (3.1)$$

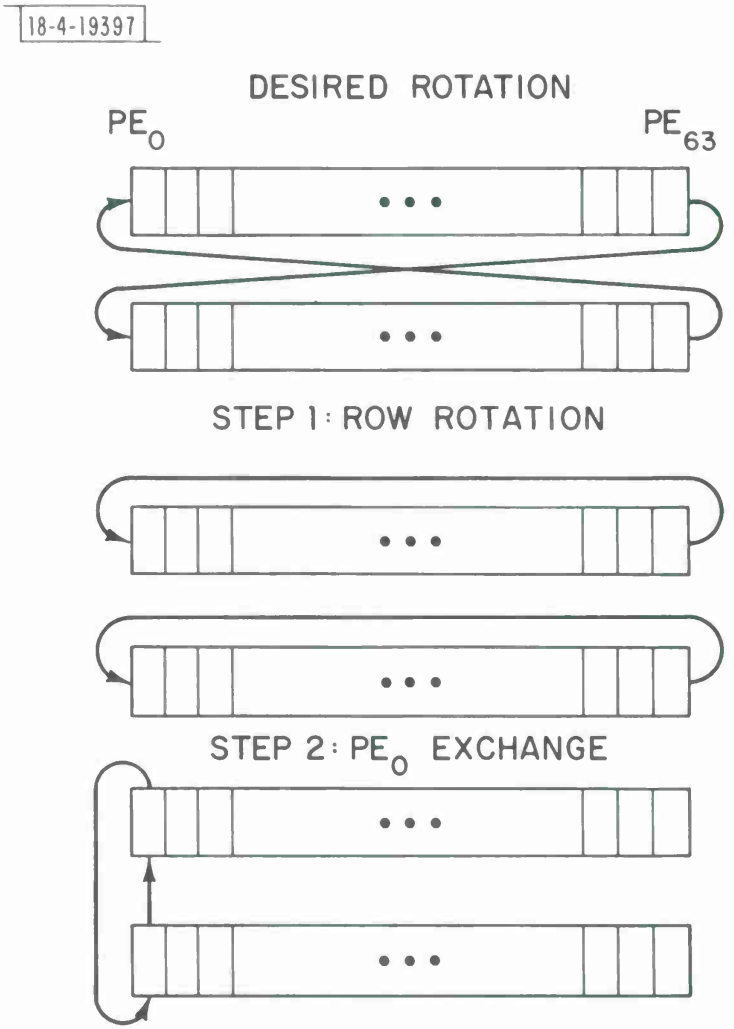


Fig. 3. Cyclic Rotation Modulo 128 on Illiac.

where C is the cycle time in nanoseconds. This working expression is plotted in Figure 4.

The code for the Illiac was prepared in GLYPNIR, an ALGOL-based language with no automatic optimization features. Thus, although assembly language listings were used to minimize unnecessary overhead, no effort was expended to introduce any arithmetic chaining since that would have required extensive machine dependent code. Furthermore, the timing was done with the clock rate set at approximately 80 nanoseconds per cycle. Thus, the measured time of 9 milliseconds for the approximately 83K floating point operations corresponds to a raw rate of 9.2 MFLOPS which, corrected for the 85% PE utilization is equivalent to 10.8 MFLOPS of full PE utilization versus 20 MFLOPS achievable with no chaining or 54% of the maximum achievable arithmetic rate. Assuming that the 46% overhead is retained, but that chaining is introduced to an effective chain length of N = 3 on the average, then the achievable Illiac rate would be 18.4 MFLOPS for C = 80 nanoseconds or 29.4 MFLOPS for the design value of C = 50 nanoseconds.

C. The CDC-STAR Algorithm

The pipeline architecture (see [49], [51], [52]) is unconstrained by small fixed resources (i.e., 64 processors). On the other hand, efficient utilization of the pipeline requires detailed attention to prearrangement of vectors to allow for streaming from consecutive memory locations. This consideration is particularly important for STAR, which has a relatively slow memory cycle time. Since the nonlinear filter is recursive, it is necessary to include the vector rearrangement as part of the filter update and therefore the rearrangement constitutes the major overhead of the STAR program. The operations on the matrix F_{n-1} to precondition it for the convolution are shown in Figures 5 and 6. First, the column-ordered F_{n-1} matrix is column-shuffled with itself to produce a matrix which has two copies of every phase variable in each column. Then, a scrambled F_{n-1} can be formed which has the property that each row in the final convolved matrix can be generated by operating on a suitable interpolation between the two adjacent rows of the scrambled F_{n-1}.

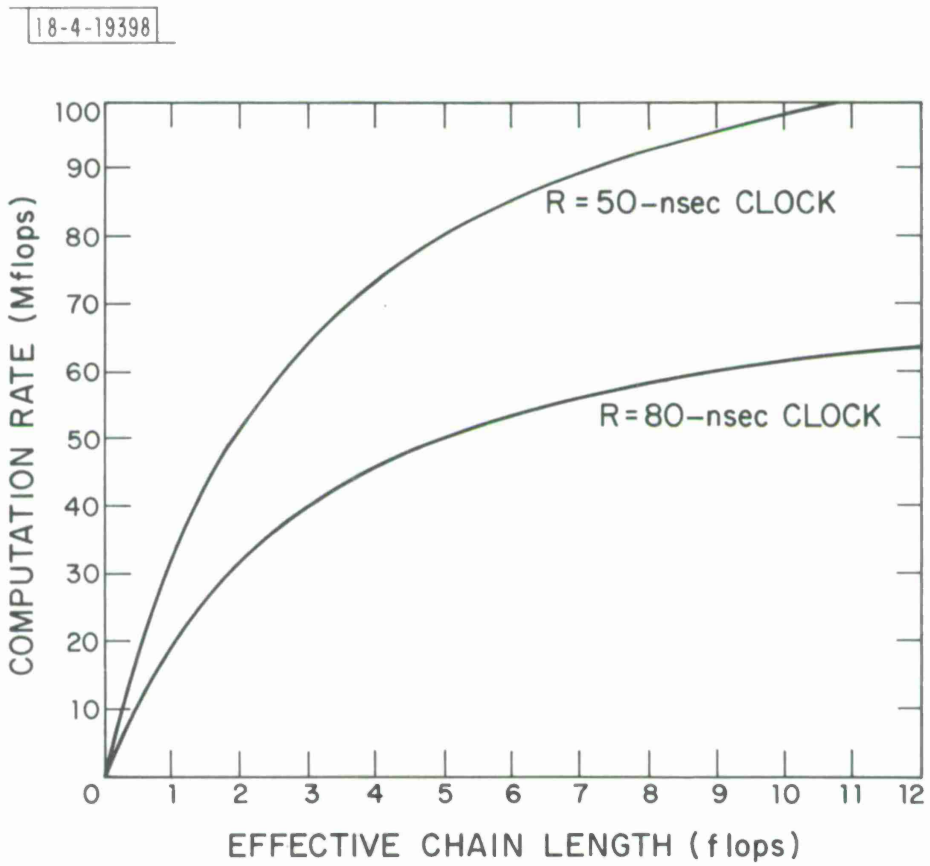


Fig. 4. Maximum computation rates for Illiac IV.

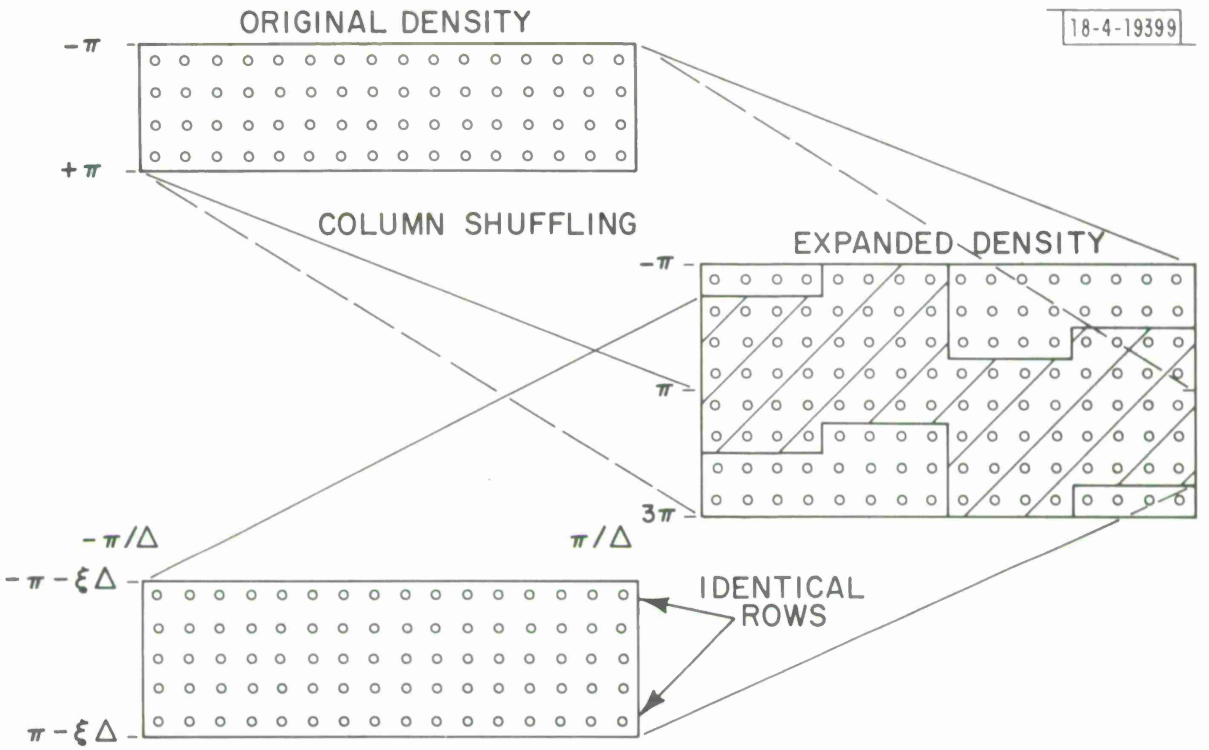


Fig. 5. Scrambling of phase variables modulo 2π.

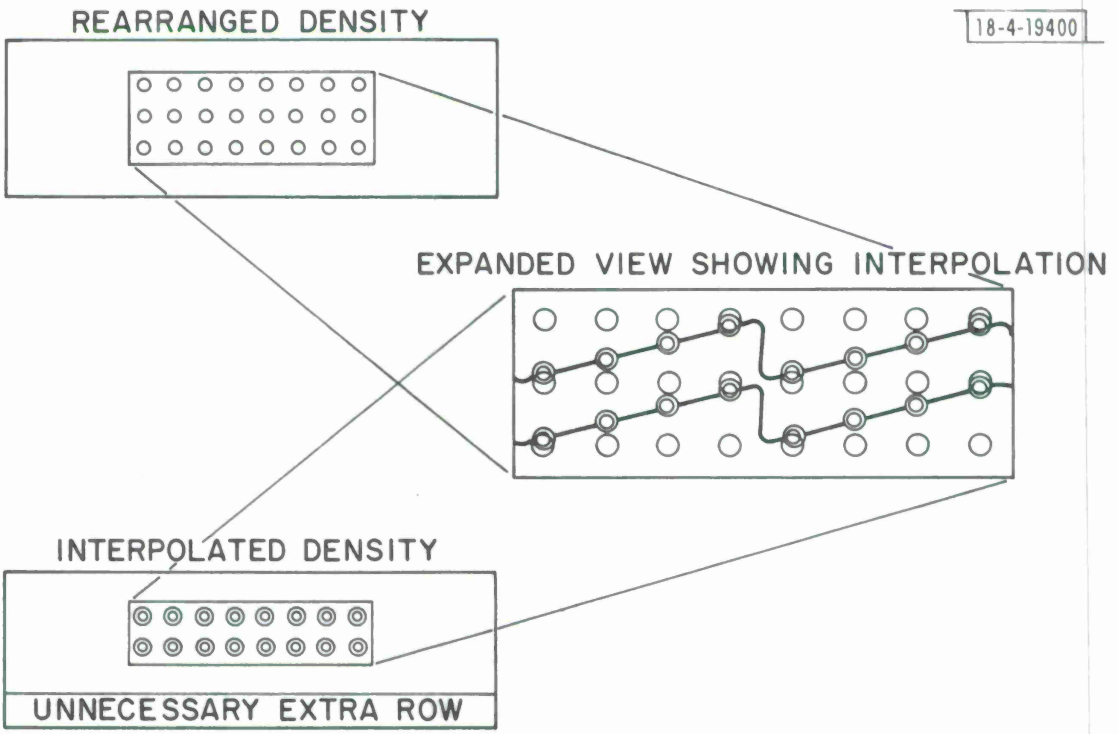


Fig. 6. Interpolation of scrambled matrix.

The interpolation which does this is depicted in Figure 6*. This interpolation may be done by vector operations of length (m+1)n, or 4224. The row corresponding to m+1 in the result may be compressed out of the final result to reduce the subsequent calculations.

The cyclic convolution is shown in Figure 7. First, the end columns of the interpolated F_{n-1} are cyclically copied to produce a matrix from which each of the terms of the convolution may be obtained as m x n matrices. The production of a 5-term symmetric convolution is done in parallel for all 4096 points by a sequence of 10 vector adds and 5 vector multiplies, all of length 4096.

The only computations which are done on the STAR that are less than 100% efficient are the vector sums and the determination of the estimates. This is reflected in Table 2, which gives the breakdown of the various functions in the STAR program.

The measured execution time of 5.17 milliseconds for the STAR version of the filter corresponds to an overall processing rate of about 16 MFLOPS. The rate for required arithmetic only (only 63% of the total time) would be 25.3 MFLOPS. Now the STAR is capable of producing vector sums at a 50 MFLOPS rate and vector multiplies at the 25 MFLOPS rate and, from Table 1, we find that 64.9% of the floating point operations required are additions and 35.1% are multiplies. Furthermore, although the ratio of compute to compute plus start-up cycles in Table 2 suggests 95.6% efficiency for multiplies and 90.4% efficiency for additions, when account is taken of the unnecessary adds and multiplies introduced in order to force vector arithmetic, the asymptotic computation rates for this problem are corrected downward to 16.8 MFLOPS and 39.1 MFLOPS for multiplication and addition, respectively. On the whole, however, if the running times are corrected down to the estimated minimum achievable 4.70 milliseconds then the arithmetic results would be produced at

*The Figures 2 and 3 are shown with m=4 and n=16 for illustrative purposes only.

18-4-19401

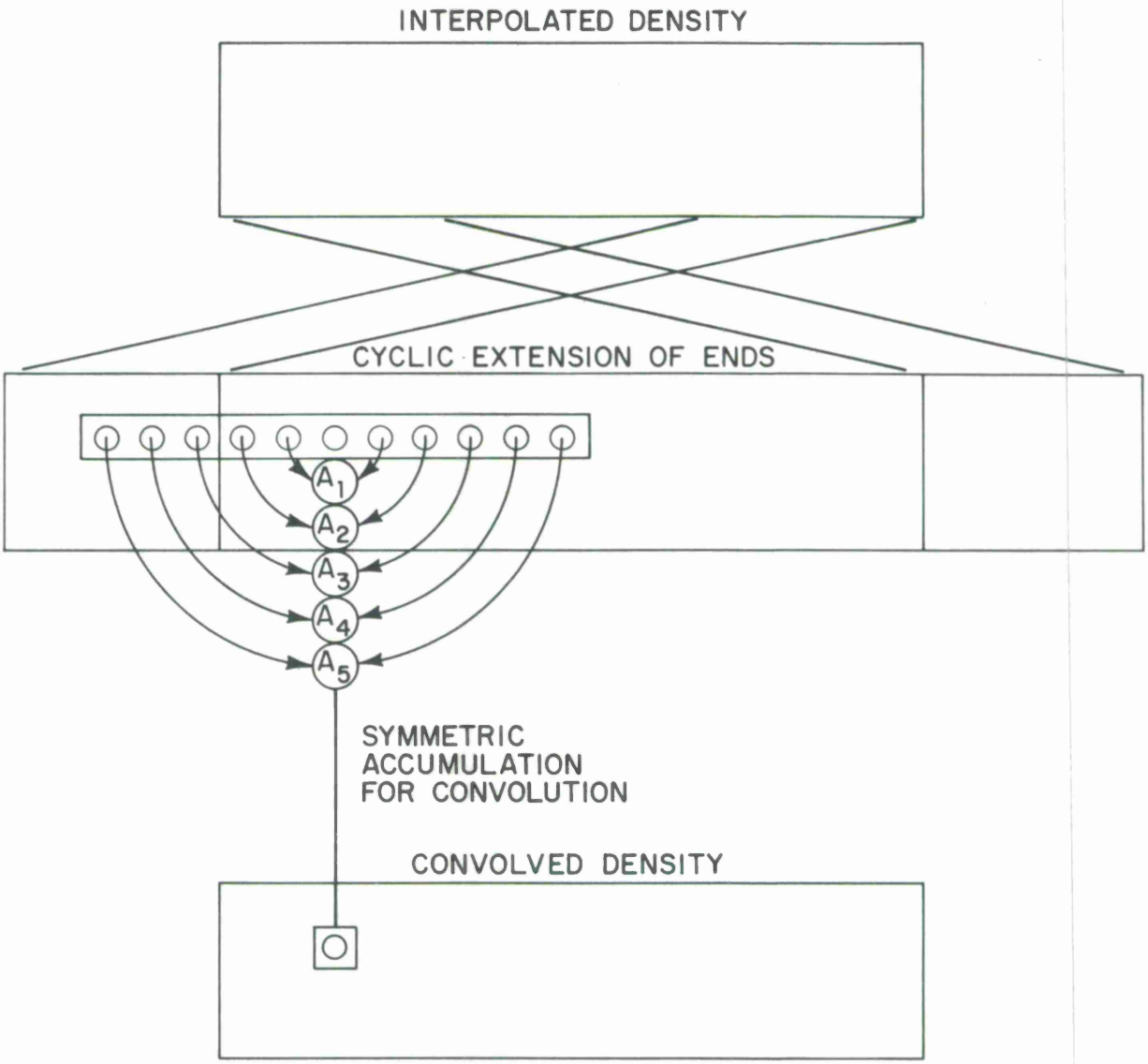


Fig. 7. Convolution operation.

TABLE 2

STAR PROGRAM BREAKDOWN

Operations	% of Total	Cycles		
		Start-ups	Compute	Required
Vector Arithmetic	63.4	5174	76793	78.5%
12 Multiplies	33.3	1908	41152	70.1%
46 Adds	26.4	3266	30861	86.4%
1 Exponential	3.7	--	4780	100.0%
Vector Rearrangement	31.3	1233	39292	83.7%
1 Vector Transfer	5.8	1001	6464	68.0%
2 Indexed Transfers (Block lengths 32 & 33)	22.1	144	28480	100.0%
1 Vector Compress	3.4	88	4348	0%
Scalar Arithmetic	--	--	79	100.0%
1 Divide			46	
3 Adds			33	
Subroutine Overhead	4.1	--	5142	0%
3 Calls				
Miscellaneous	1.2	--	1537	100.0%
Memory Conflicts, etc.				
TOTAL		6407	122843	5.17 msec
		5%	95%	
Minimum Achievable		6224	111296	4.70 msec

a rate which is 52.1% of the asymptotic rate for this problem (i.e., without counting overhead or unnecessary computations), and, discounting vector rearrangement, 77.1% of the asymptotic rate. We conclude, therefore, that the nonlinear filter utilizes the STAR about as efficiently as could be expected for any practical problem.

The CDC STAR was programmed in a vectorized form of FORTRAN IV with the use of assembly language listings to aid in optimization. It was necessary to insert two inline machine instructions to implement the vector block transfers for the rearrangement process. These instructions were not as yet supported by the FORTRAN system. Moreover, we were compelled to replace one machine instruction which was supported (the SUM instruction, which accumulates the sum of components of a vector) by an entire vector subroutine in order to increase execution speed. Since these changes were anything but obvious we view the language support to be somewhat deficient for this particular exercise, although clearly not so difficult to use efficiently as the Illiac IV language.

D. CDC 6600 Program

The CDC 6600 (and 7600) has instruction look-ahead and multiple arithmetic functional units which provide a partial overlap parallelism. The most efficient 6600 programs contain many tight loops instead of complicated computations within large loops. By recoding the vectorized STAR program in analogous FORTRAN for the CDC 6600, we were able to achieve efficient utilization of the available resources with functional loops which are for the most part contained within the 8-word instruction stack.

The basic data flow of the CDC 6600 is illustrated in Figure 8. Reads from memory are accomplished by setting the A Registers A1-A5 with the appropriate address; writes are obtained by loading A6-A7. The B Registers are used for incrementing and address computation.

The breakdown of the computations for the CDC 6600 is shown in Table 3. Note that the major overhead is for reading and writing and miscellaneous waiting. It is interesting to note also that the ratio of multiply rates

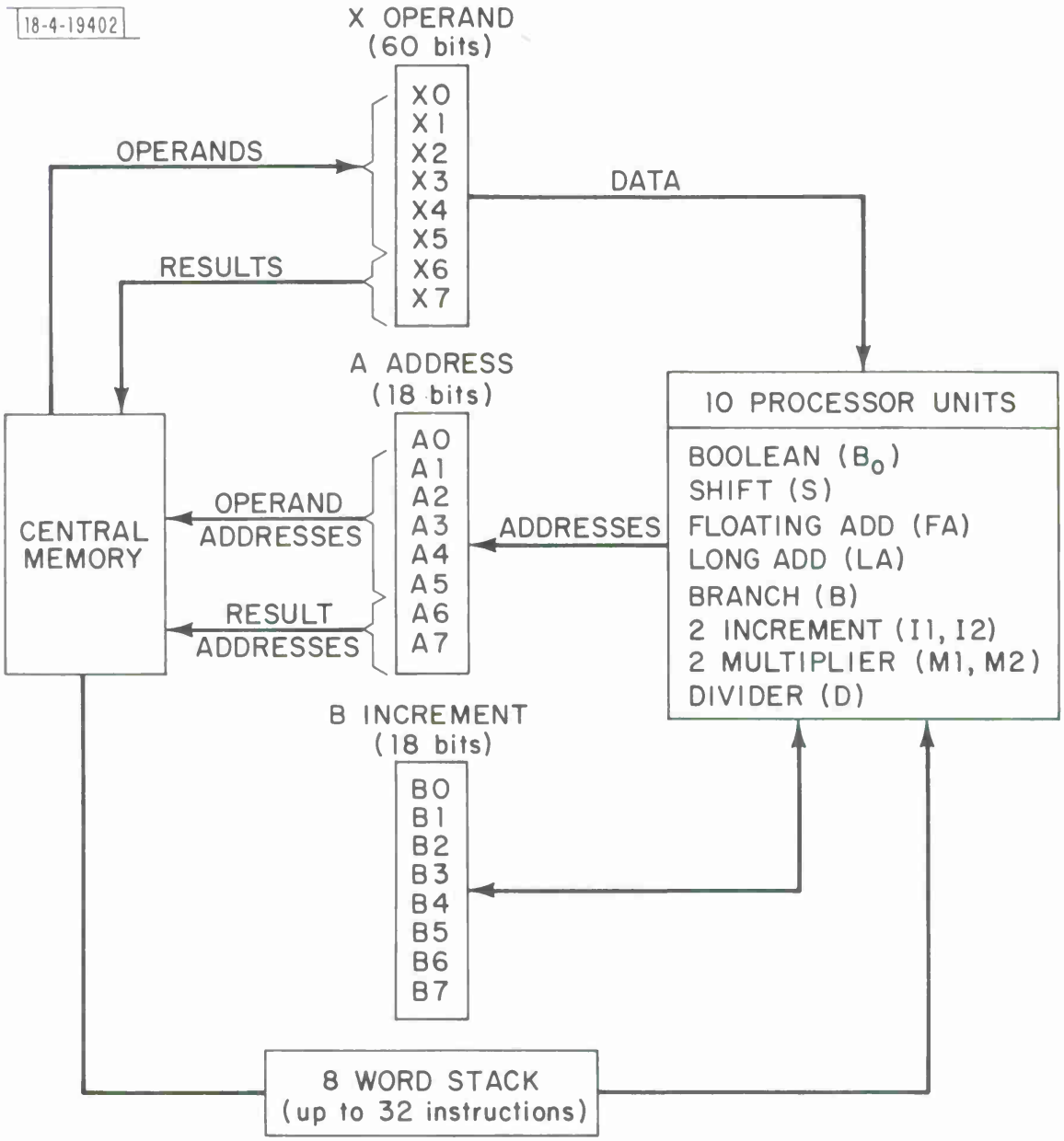


Fig. 8. CDC 6600 CPU architecture.

TABLE 3

CDC6600 PROGRAM BREAKDOWN

Function (% Cycles)	Multiplies	Adds	Divisions	Reads	Writes	Exp.	Notes
Sensor Update (~0)	2m	m	0	3m	m	m	Not in stack (extn. refs.)
Interpolation (~16)	nm	2mn	0	4nm+3n	mn	0	Outer loop not in stack
Expansion (~0)	0	0	0	10m	10m	0	Instack
Convolution (~74)	5mn	10mn	0	16mn+15	6mn	0	Instack
Row Sums (~4)	0	mn-m	0	mn+m	m	0	Instack
Estimates (~0)	3m	3m-3	1	4m+11	3	0	Instack
Normalization (~6)	mn+m	0	0	mn+2m	mn	0	Instack
Approx. Totals	7mn	13mn	0	22mn	8mn	0	
Approx. No. of Minor Cycles	70mn	91mn	0	66mn*	24mn*	0	

Summary: Minor Cycles (Approx.)

Arithmetic:	161mn	47.1%
Read/Write:	90mn	26.3%
Waiting:	91mn	26.6%
Measured:	342mn	100.0%

(m = 32, n = 128 for test case) Measured time = 140 msec/est.

*The reads and writes are partially overlapped in time, however the total overhead of 52.9% is accurate.

between STAR and 6600 is 25 to 1, while the add-rate ratio is 35 to 1 (including normalization on 6600). Thus, for arithmetic alone, STAR would be expected to be 31 times faster than the 6600 on this problem. The achieved speed-up of 27 is therefore reasonable.

The CDC 6600 was programmed in standard FORTRAN IV for this problem. It was necessary to code all two-dimensional arrays as one-dimensional arrays and to make several iterations with assembly language listings produced by the FTN 4.6 level 428 optimizing compiler before the final running time of 130 milliseconds per estimate, which corresponds to 0.63 MFLOPS, was achieved. It should also be acknowledged that the present 6600 program does not make use of the second multiplier, since to do so would force the affected loops out of the stack. In fairness to the 7600, however, which has a larger stack, we might be able to increase the 7600 running speed by as much as 10% by using both multipliers. Nevertheless the CDC 6600 and 7600 serve as good benchmarks for comparison of achieved efficiencies and software development difficulties. The three-dimensional problem is intractable for the 6600, however, so no sttempt has been made to characterize its performance. We are currently studying the three-dimensional problem on the 7600. This problem requires array storage in excess of 65K, which leads to complicated memory management considerations.

E. The AP-120B Algorithm

The Floating Point Systems AP120B architecture (see [47] and [55]) is illustrated in Figure 9. There are sufficient data paths to permit a number of essentially independent operations to proceed in parallel. Microcode for the AP120B is included in 64-bit microinstructions, as shown in Figure 10, which are programmed and cross-assembled in a Macro assembly language for a host mini-computer. Although the software development for the AP120B must be considered tedious by comparison with the larger machines, the AP120B can nevertheless provide competitive performance at very low cost. Thus, we have optimized a two-dimensional example for the AP120B to show its performance in the most competitive light, even though we have not expended equivalent effort in optimization for other machines.

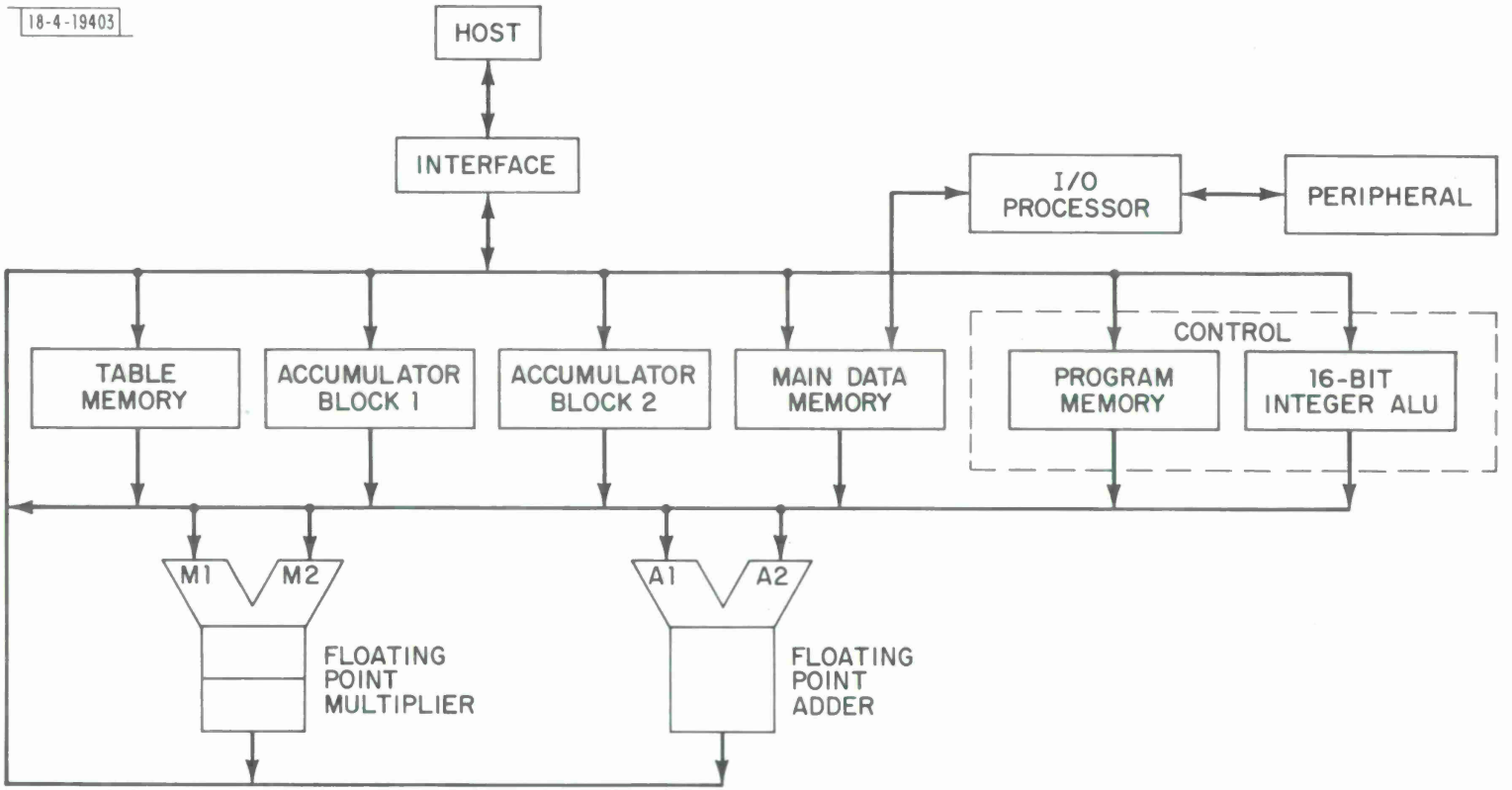


Fig. 9. AP-120B processor block diagram.

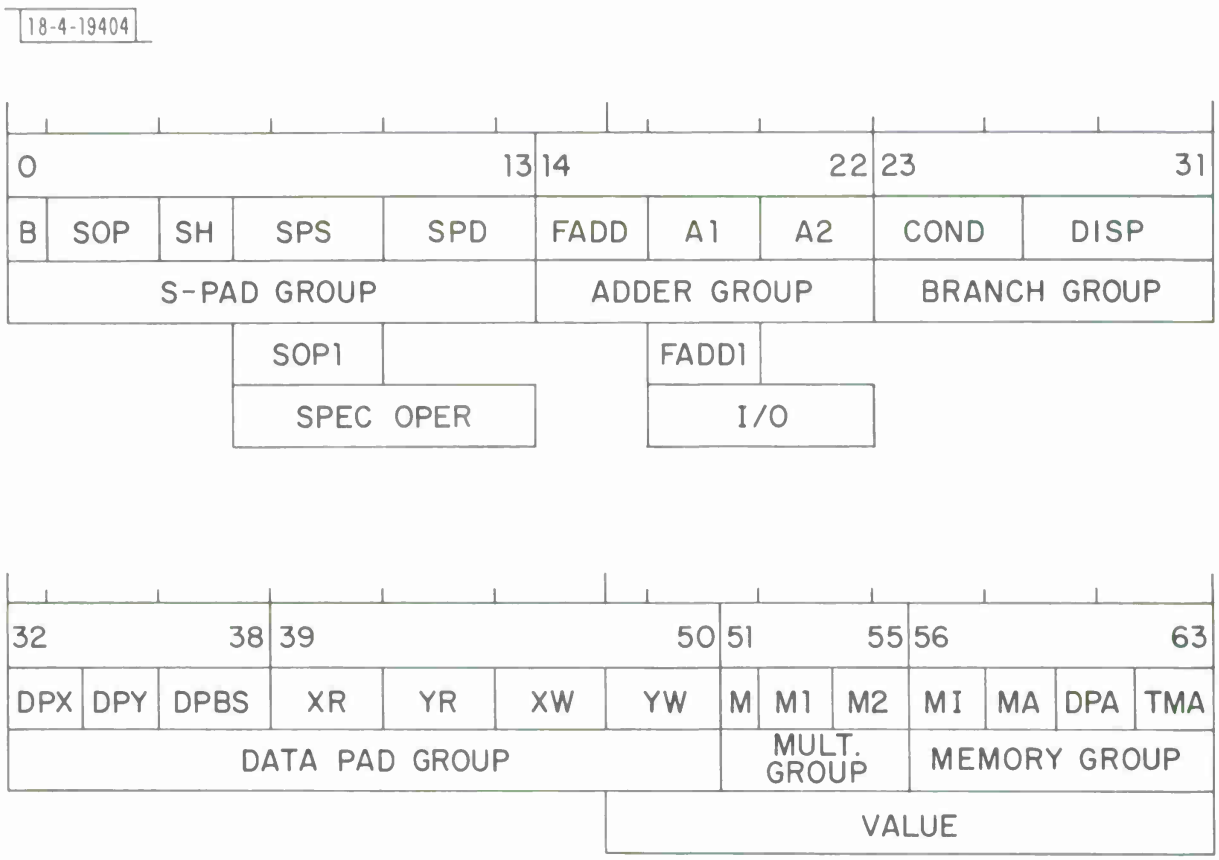


Fig. 10. AP-120B microcode format.

The initial code for the AP120B was derived by translating the CDC 6600 FORTRAN IV code described in the previous section. The only modification introduced was to implement the convolution in a one-dimensional pipeline, as suggested by Randy Cole of USC/ISI. This original program required 22.8 msec per estimate and utilized three times as much memory as was required to store the density information. An alternative formulation, taking into account the intrinsic odd/even memory paging constraints, was eventually developed, resulting in a computation time of 13.88 msec per estimate. This latter, essentially minimal, program is the subject of the present discussion.

The various conditional densities can be visualized as 32 x 128 matrices of weights representing the appropriate probabilities, with the (i,j)th weight associated with the ith phase and jth phase rate weight, even though in the machine this matrix is stored as a 4096-element vector made up of the ordered columns of the matrix. The code which realizes update of the one-step predictor $P_{n-1} \rightarrow P_n$ is broken into two parts $P_{n-1} \rightarrow F_{n-1}$ and $F_{n-1} \rightarrow P_n$, which are respectively column-oriented and row-oriented operations.

We describe first the passage from the predictor to scrambled filter density, $P_{n-1}(x-y) \rightarrow F_{n-1}(x-\Delta y, y)$. Assume that P_{n-1} has been stored unnormalized (i.e., not divided by K_{n-1}). Then, $(1/K_{n-1})\ D_{n-1}(x_i)$ is written on ith place of both Data Pad X (DPX) and Data Pad Y (DPY) (See Fig. 9). Next, starting from the last column of P_{n-1} the column elements are written alternately on DPX and DPY in the positions corresponding to their order in the column, starting with DPX or DPY depending upon whether the largest α with $x_i - Dy_j \geq x_\alpha$ is odd or even for the jth column. This interweaving of the values of a column of P_n is necessary because the AP120B microcode permits at most one access to each of DPX and DPY in one instruction. Thus, for the jth column, $P_{n-1}(x_\ell, y_j)$ in DPX(ℓ) and $P_{n-1}(x_{\ell+1}, y_j)$ in DPY(ℓ+1) are replaced by $(1/K_{n-1})\ D_{n-1}(x_\ell)$. $P_{n-1}(x_\ell, y_j)$ and $(1/K_{n-1})\ D_{n-1}(x_{\ell+1})\ P_{n-1}(x_{\ell+1}, y_j)$, respectively, while $P_{n-1}(x_{\ell+2}, y_j)$ and $P(x_{\ell+3}, y_j)$ are read from main data memory into DPX(ℓ+2) and DPY(ℓ+3). Next, the interpolation of the contents of DPY(ℓ-1) and DPX(ℓ) are written into $P_{n-1}(x_{\ell-\alpha}, y_{j+1})$ and $P_{n-1}(x_{\ell+1-\alpha}, y_{j+1})$, respec-

tively. During the above operations $\ell = \alpha + 1, \alpha + 3, \ldots, \alpha + 31$ and these indices are interpreted modulo 32 by masking out all but the last 5 bits. We have described the situation when α is even; if α is odd the DPX and DPY are interchanged. The result of the $P_{n-1} \rightarrow F_{n-1}$ operation, the scrambled values of F_{n-1} reside in main data memory overlapping the old P_{n-1} but shifted to the right by one column (i.e., 32 locations).

The next part of the update consists of the convolution of the scrambled values of F_{n-1} with a gaussian kernel to produce P_n. Because of the mathematical structure of our problem the convolution task is row-oriented; that is, 32 one-dimensional convolutions, one for each row of the scrambled filter density.

We will illustrate the convolution for a general problem where the convolution kernel a_i has finite support, i.e., $a_i = 0$ for $|i|>5$. Denote k_i as the input sequence and ℓ_i as the convolved output, then

$$\ell_j = \sum_{i=-5}^{5} a_i \, k_{j-i}$$

This noncausal tapped delay line is implemented as a pipeline (see Figure 11) within the registers of the AP120B as follows: ten registers are used to store the partial convolutions for ten consecutive values of j. Then the k_i are inserted serially into the pipeline.

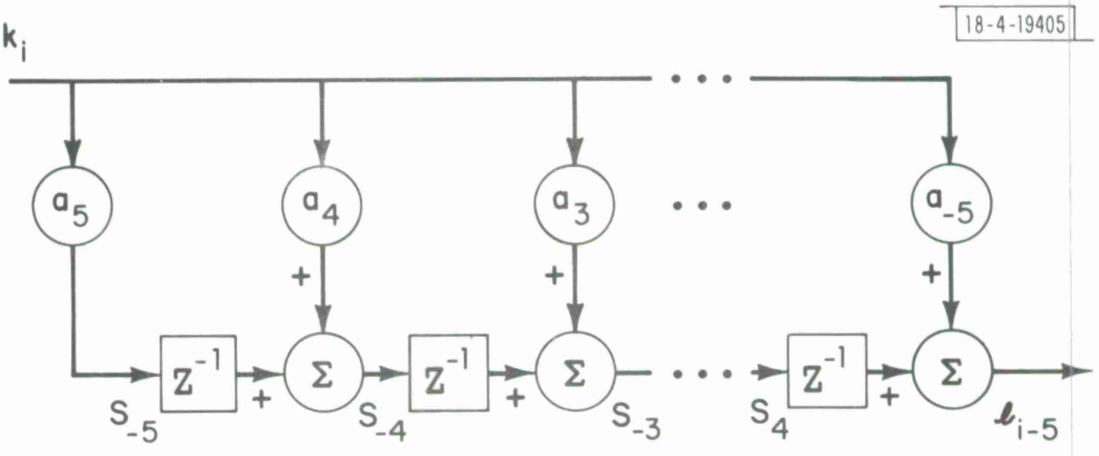


Fig. 11. Convolution flowgraph for AP-120B.

The pipeline is charged up according to the sequence below, which produces ℓ_1 at the output:

Step	Computed Terms				Instructions	Flops
1	$k_{-4}a_5$				1	1
2	$k_{-3}a_5$	$k_{-3}a_4$			2	3
3	$k_{-2}a_5$	$k_{-2}+a_4$	$k_{-2}+a_3$		3	5
	...	+	+	+		
11	k_6a_5	k_6a_4	k_6a_3	$\cdots k_6a_{-5}$	11	21
					66	121

Thereafter, until ℓ_{n-10}, ℓ_i is produced in 11 instructions with 21 FLOPS (11 multiplies and 10 adds). After k_{n-1} is inserted into the pipeline, then k_0, k_1, ..., k_5 must be inserted again to complete the calculation. During the last 10 iterations the pipeline may be shut down in a manner analogous to the above build-up, so that the last 10 ℓ_i are produced in a total of 55 instructions, or 100 FLOPS. Thus, in summary we can produce 128 convolution terms with 110 + 118 (11) = 1408 = 11 (128) instructions and 200 + 118 (21) = 2678 FLOPS, for an overall arithmetic efficiency of 95%. On the other hand, if we note that $a_i = a_{-i}$ and $a_o = 1$, we see that the total required computations for this operation equals 128 (10 ADDS + 5 MULTIPLIES) = 1920 FLOPS, so that in reality the convolution cannot be made anymore than 68% efficient, since 1408 instructions will ideally permit 2816 FLOPS. The fact that AP120B achieves 5.9 MFLOPS, then, instead of the theoretical 12 MFLOPS, is partially explained on the basis of unnecessary arithmetic operations which are introduced to produce the fastest overall computation speed and to code a general convolution loop for nonsymmetric kernels. This is analogous to the vectorization of some functions on the STAR wherein superfluous computations were introduced in order to achieve the minimum overall computation time.

AP120B Code for the 3-D Problem

The software for the 3-D Problem used the described 2-D code as a subroutine as follows: $P_n(x,y,z)$ for each fixed amplitude z is first updated by the 2-D microcode described above with sensor density different for different z to obtain $\tilde{P}_{n+1}(x,y,z)$, x and y take values on a fixed 16 x 96 grid while z has value on a moving grid centered at the conditional amplitude mean and with mesh proportional to the conditional amplitude standard deviation.

The conditional distribution $P_{n+1}(x,y,z)$ is obtained from $\tilde{P}_{n+1}(x,y,z)$ by convolving over the z direction. See [10], [12], [15], [18], [19], for details on how convolution is performed on a moving grid. This final convolution is achieved by a software pipeline as in the case of the previous convolution. The validity of separation of the update in this way depends on the statistical independence of the amplitude and phase processes.

F. The CRAY-1 Algorithm

The CRAY-1 (see [48], [53], [56]) represents a logical extension of many of features of the CDC6600 system. It may be viewed in a global sense as a multifunctional unit machine with lookahead scheduling. On the other hand, closer inspection reveals a number of innovations which extend the generality of the design and which go a long way toward eliminating unnecessary constraints (see Fig. 12). Probably the most significant improvement is to provide segmentation of all functional units so that functional unit reservation need only wait until the input and output registers are free, independent of the amount of time required to compute the function. In addition, two new features significantly impact the applicability of the CRAY-1 for vector processing. First there are sight vector registers which may be filled by any set of operands composing any linear vector in main memory* of length less than or equal to 64. The vector registers may be used to provide inputs or outputs to any of the floating point functional units at an 80 megaflop rate. Second, and even more significant, the output results from one vector operation can be

*Memory bank access conflicts only occur if the consecutive operands occupy memory locations separated by a multiple of 16 locations.

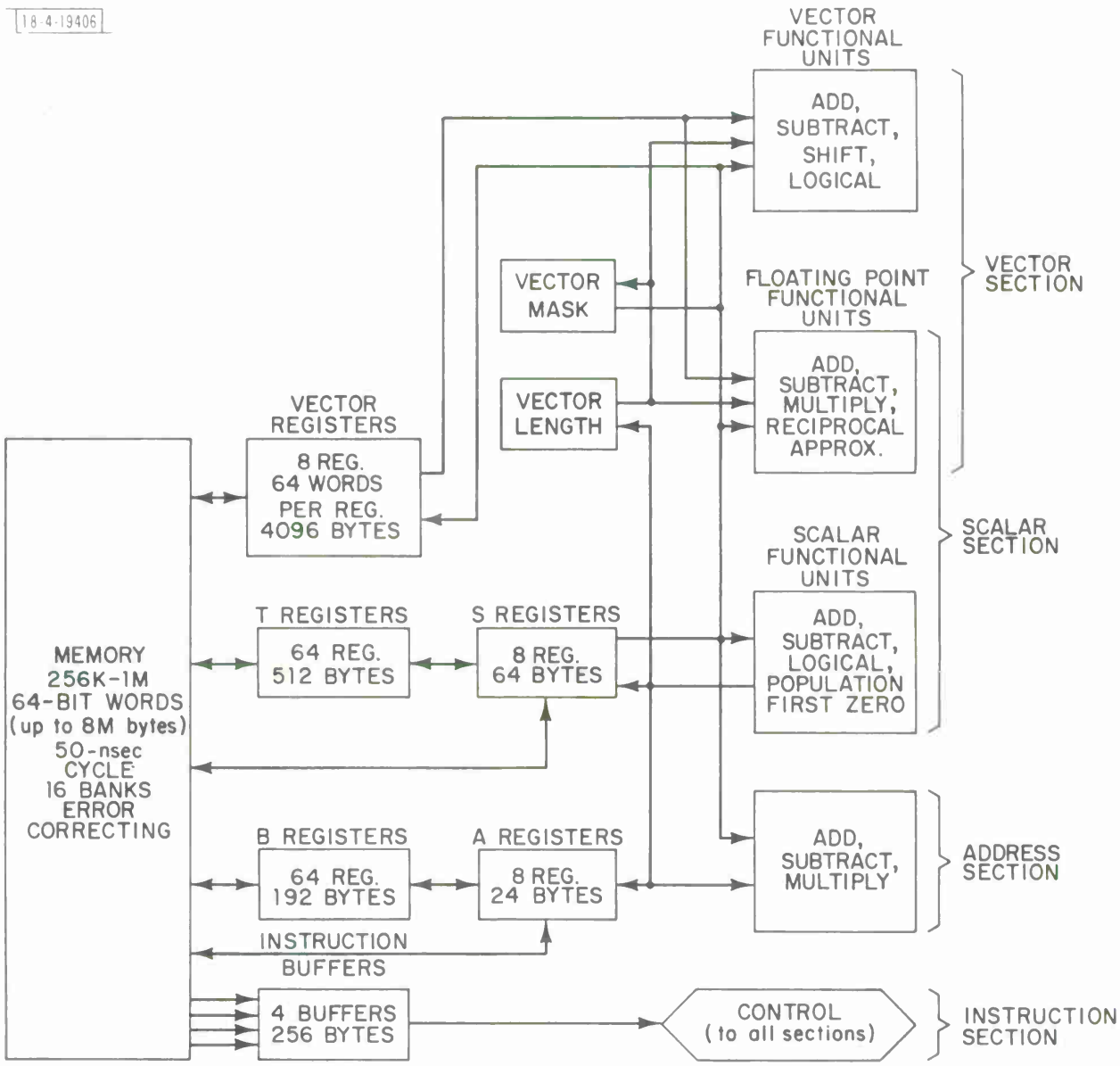


Fig. 12. CRAY-1 register block diagram.

chained with inputs from a third vector register as they appear, thus avoiding a second start-up delay. Thus, with addition chained with multiplication, the CRAY-1 is asymptotically capable of producing up to 160 megaflops.

The architecture of the CRAY-1 therefore has considerable potential. What we were concerned about was the accessibility of this to the potential user. We are encouraged to report at this time that it appears that a truly no-nonsense exploitation of this resource is readily available. In fact, we were able to write a very simple program in standard FORTRAN-IV which was able, with only minor modifications, to achieve over 22 megaflops for the demodulation problem. The simplicity of this program is characterized by the following code which produces a convolved matrix JN from an interpolated and cyclically rotated matrix JN1:

```
      DO 40        I = 1, 32
      DO 35        J = 1, 128
35    JN(I,J)  =   JN1(I,J+10)
      DO 40        K = 1, 5
      DO 40        J = 1, 128
40    JN(I,J)  =   JN(I,J) + A(K)*(JN1(I,J-K+10)+JN1(I,J+K+10))
```

What is noteworthy about this example is that the first generation CRAY-1 compiler is capable of isolating the linear vectors JN(I,.) and JN1(I,.), where the column index is a constant offset (in the inner loop) from the inner loop index J. Furthermore, the compiler is capable of using addition and multiplication chaining in line 40. The only inefficiencies introduced by the compiler in this example are associated with loop set-up and control.

Since the results obtained to date on the CRAY-1 indicate considerable promise for the development for vector-optimized compilers, we feel that the CRAY-1 architecture is very likely to be very attractive as a model for versatile, general-purpose computers of the future. In particular, the use of multiple, segmented functional units and a variety of general purpose registers appears to provide desirable flexibility.

IV. EXPERIMENTAL RESULTS

For the 2-D phase demodulation problem, extensive Monte Carlo runs have been made at a variety of input signal to noise ratios, and the phase error modulo 2π for the phase lock loop and the optimal cyclic estimator have been evaluated. The results of these Monte Carlo runs are given in Fig. 13, where each point on the optimal curve resulted from averaging the squared errors of three million estimates with three-standard-deviation confidence intervals of $\pm$ .034 dB about each point. The experimental design consisted of producing 30,000 independent sample paths, with each path consisting of 130 samples in time of the phase error mod 2π. The first 30 errors were discarded as we were interested in steady state performance. These results were obtained using the AP120B array processor in conjunction with the PDP-11-55 with the multi-user operating system RSX11M version 3. Each point on the curve represents two days of computer time. Of this total one day represents overhead of the the operating system of the PDP-11. The other day is the array processor time. The array processor is achieving a little more than one quarter of its theoretical 12 megaflops (i.e., millions of floating point operations per second). New software for the array processor has been designed which achieves 5.9 megaflops. For these experiments a fixed grid of 32 x 128 was used (i.e., 32 subdivisions in phase and 128 in phase rate). The resulting performance curve Fig. 13 should be compared with the analogous curve in [32] where the grid was 21 x 105 and 200 sample paths were used to find the error.

The problem of combined phase and amplitude demodulation was first considered in a program for the STAR-100 at NASA Langley and is described in [45]. Originally with H linear (see (2.5) and (2.6)) for a number of output signal to noise ratios, the abssica of Fig. 13, with the amplitude being normal mean 1 and variance .05 and generated by (2.6), Monte Carlo runs showed that unknown amplitude reduces performance only slightly. When H is exponential and $\ln H(A_n)$ has variance 0.1 again performance closely resembles the fixed amplitude case. We are currently investigating the case where $\ln H(A_n)$ has variance about 2 and H is exponential, as this seems to be a physically

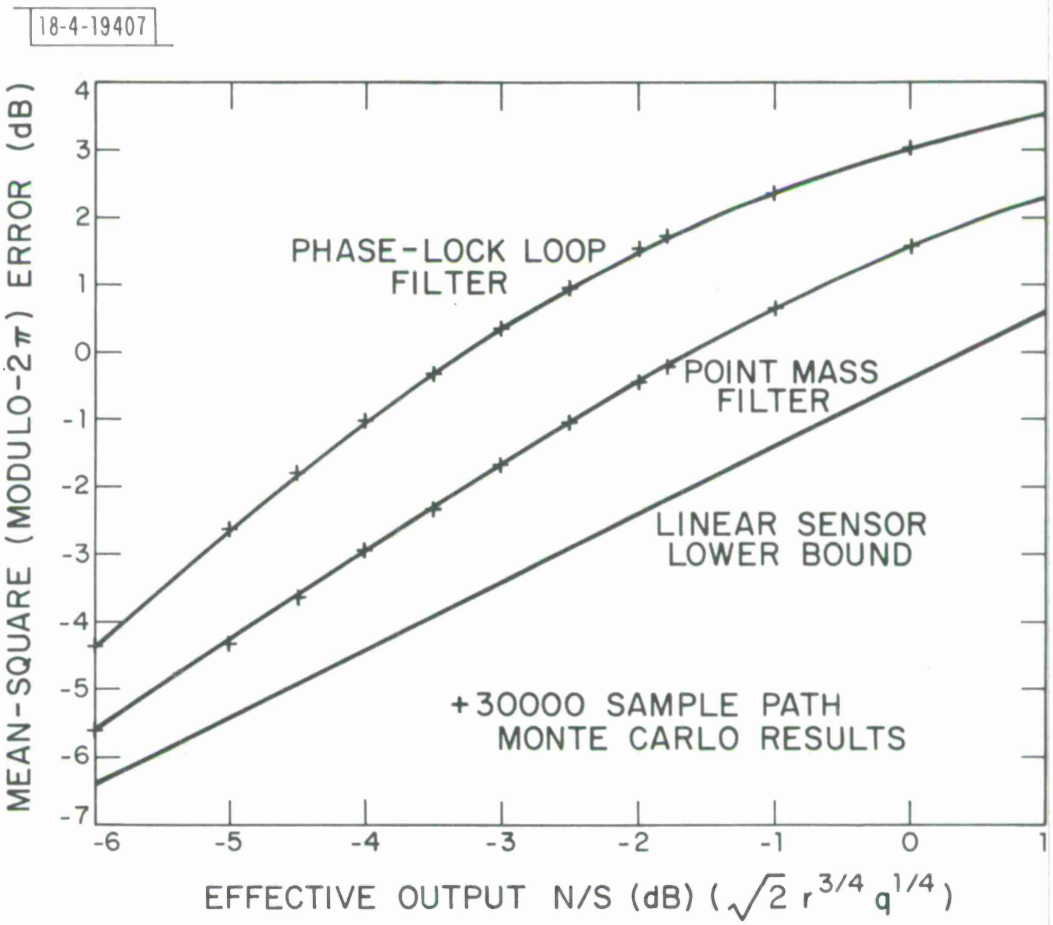


Fig. 13. Monte Carlo performance summary

interesting case. At the moment assembly code is being developed for the 3-D phase demodulation problem using the AP120B. For the 3-D problem we expect less impact from the PDP11 system overhead because a larger proportion of time will be spent in the AP120B.

In Table 4, the results of ongoing timing studies for various machines for the 2D phase demodulation problem are given. In the case of each machine extensive considerations of the blending of algorithm structure to fit machine architecture were used to develop fast code. Generally the software seems to fit into three groupings; 1) parallel; Illiac 2) one big main loop; CRAY-I and 120B; 3) a large number of small loops comprising the main loop for 6600, 7600 and STAR-100. Categories 2) and 3) above are illustrated in Figures 14 and 15 which show the software division in the 2-D phase demodulation algorithm, each box represents a multi-loop. Category 1) is discussed in the section describing the Illiac coding. It is apparent that assembly language coding could produce faster times for the larger machines with a corresponding increased software development time.

In reviewing the results of Table 4, it is possible to draw some general conclusions. On the one hand we see that the vector architectures can achieve substantial increase in speed, resulting in correspondingly lower facility cost per performance. On the other hand there is in general a corresponding increase in software development time for the vector machines, so that if experimental code is desired it may turn out to be just as expensive to develop and run on a vector machine as on a conventional machine. There are two obvious standout cases, however, which deserve special mention: for the AP120B the observation is extremely low cost (by an order of magnitude) for performance, which implies low production cost; the CRAY-1, with its FORTRAN compiler, may turn out to be the best of both worlds when experimental code is being both written and tested. Moreover, as the problem dimension increases the practicality of the AP120B for extended Monte Carlo analysis will be diminished, since, for example, we estimate that 23 days of continuous running

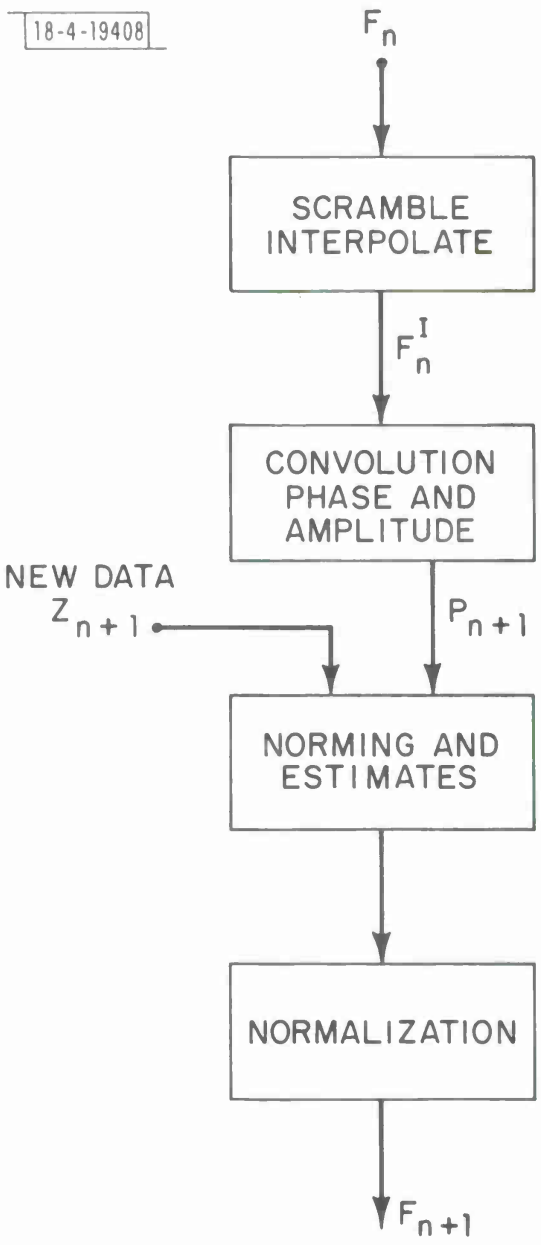


Fig. 14. Recursive filter

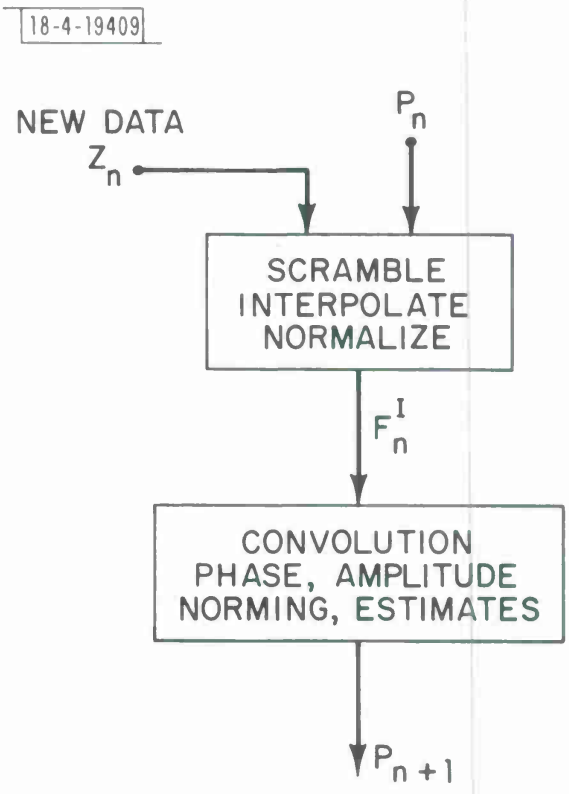


Fig. 15. Recursive predictor

TABLE 4

COMPARATIVE PROCESSOR

COST/PERFORMANCE FOR DEMODULATION PROBLEM

Machine	Time per Iteration (msec)	Achieved Megaflops*	Approx** Cost (Dollars/ Flop)	Max Theory Megaflops	Software Develop. Time (man-months)
CRAY-1	3.5	23.5	0.33	60-140	0.5
STAR-100	4.9	16.8	0.48	20-40	2.0
Illiac IV	9.0	9.1	1.10	40-80	3.0
AP120B	13.9+	5.9	.03	12	6.0
CDC7600	25.0	3.3	.91	10	1.1
IBM370-168	110.0	0.75	2.67	3	1.0
CDC6600	130.0	0.63	1.59	2	1.0
PDP11-70	870.0	0.09	1.67	0.2	1.2

*Assumes 82.2K Flops per iteration.

**Assumed installation costs of (production) systems:
CRAY-1 - $8M, STAR-100 - $8M, Illiac-IV - $10M, AP120B - $150K,
CDC7600 - $3M, IBM370-168 - $2M, CDC6600 - $1M, PDP11-70 - $150K.

+Does not include PDP11-55 overhead.

time would be required to produce the four million estimates for the Monte Carlo analysis of the 3-D phase demodulator. Thus, even with restartable software, such research would push the reliability limits of the AP120B. It is clear, then, that the CRAY-1 and its descendents will usher in a new generation of research computer performance, while the AP120B will provide a prototype for low-cost dedicated-applications computation.

V. CONCLUSIONS

In this paper we have presented a summary of our results on the phase demodulation problem, both for the three dimensional and two dimensional cases. In the latter case extensive and definitive Monte Carlo error analysis runs have been made for both the phase-locked loop and the optimal phase demodulator. These results document the performance improvement achievable by synthesis of the optimal demodulator and serve as a bench mark against which prospective suboptimal designs can be judged.

Another facet of our research is the effect of various machine architectures on the speed of estimate production, and to some extent to determine the architecture most suited to our problem of phase demodulation, and hopefully more generally the one most suited to general synthesis of nonlinear filters. Our conclusions in this regard depend on price/performance. Clearly the CRAY-1 is the machine most suited for general research on examples of our problem, however economic considerations dictate the choice of the AP120B as the cost effective compromise for production runs. Thus, at the present time the AP120B can provide an inexpensive, albeit difficult, opportunity to run Monte Carlo simulations for two-dimensional nonlinear filtering problems. There does not appear to be any inherit reason, however, for precluding the possibility of the evolution of CRAY-1 type capability and software accessibility to machines of more modest size in the near future. The popularity of the AP120B illustrates the market for specialized high-speed processors as add-on boxes for minicomputers. Perhaps the evolution of minicomputer architecture in the direction of the CRAY-1 or its successors will indeed be feasible in the future. In the meantime, accessibility to the AP120B will be greatly enhanced if some form of optimizing (FORTRAN?) compiler can be developed, although it remains to be seen whether this is feasible. Also, it is likely that extensive use of the CRAY-1 for important near-term developments of nonlinear filtering realizations will be profitable.

In pursuing the machine speed comparison we devoted some time to designing software for each machine which took advantage of the architecture of each machine. So far the guidelines for developing software for each machine are empirical rules of thumb gained by some experience and some hearsay. It has been observed, for example, that the 7600's running time for our problem could be reduced 10% by assembly language coding but because of limited machine access and cost we have not been able to do this. We should acknowledge, however, that considerable insight into the computational requirements for the demodulation problem was obtained by successively transforming according to the special requirements of each of the candidate architectures.

It is apparent that nonlinear filtering problems of state dimension three or higher for the point mass seem to tax the capabilities of most of the fourth generation machines in terms of speed and memory requirements (i.e., the 65K vector length limit on the STAR-100, and the 2K P.E. memory of the Illiac, the direct reference to 65K words of memory without paging on the AP120B). It is conceivable, however, that with code developed for optimal use of the CRAY-1 we could handle a four dimensional nonlinear filter with up to 400K mass points.

ACKNOWLEDGMENTS

Several individuals have contributed significantly to the benchmarking on the various machines: for the CDC6600 Herb Spies of Eglin Air Force Base and George Shannon of Lincoln Laboratory; for Illiac IV Alan Birholtz and Gerald Marin of the Institute of Advanced Computation, Sunnyvale, CA; for CDC STAR 100 Jim Ortega, Ed Fondriat, Evert Johnson, Richard Hofler, Illona Howser, and John Knight of the NASA Langley Research Center, for CRAY-1 Lee Higbie of Cray Research, Inc.; for the AP120B Randy Cole of USC/ISI, Alan Charlesworth and Woody Wittmayer of Floating Point Systems. In addition, a number of other persons have contributed actively to the research described here, especially Jack Mallinckrodt of Communications Research, Hussein Youssef of Lockheed Aircraft, and Feramara Ghavanlou and Tom Bleakney of USC.

BIBLIOGRAPHY

[1] K. Ito, On Stochastic Differential Equations (Memoirs American Math Society 4, 1951).

[2] F. B. Hildebrand, Introduction to Numerical Analysis (McGraw-Hill, New York, 1956).

[3] A. J. Viterbi, Principles of Coherent Communication (McGraw-Hill, New York, 1966).

[4] R. S. Bucy and P. D. Joseph, Filtering for Stochastic Processes with Applications to Guidance (Wiley Interscience, New York, 1968).

[5] H. W. Sorenson and A. R. Stubberud, "Non-Linear Filtering by Approximation of the A Posteriori Density," International J. Control 8, 33-51 (1968).

[6] R. S. Bucy, "Bayes Theorem and Digital Realizations for Non-Linear Filters," J. Astro. Sci. 17, 80-94 (1969).

[7] A. H. Jazwinski, Stochastic Processes and Filtering Theory (Academic Press, New York, 1970).

[8] R. S. Bucy, "Linear and Nonlinear Filtering," Proc. IEEE, 58, 854-864 (1970).

[9] A. J. Mallinckrodt, R. S. Bucy, and S. Y. Cheng, "Final Project Report for a Design Study for an Optimal Non-Linear Receiver/Demodulator," NASA Contract NAS5-10789, Goddard Space Flight Center, Maryland (1970).

[10] K. D. Senne and R. S. Bucy, "Digital Realization of Optimal Discrete-Time Nonlinear Estimators," Proc. Fourth Annual Princeton Conf. on System Sciences, Princeton, March 1970, 280-284.

[11] K. Srinivasan, "State Estimation by Orthogonal Expansion of Probability Distributions," IEEE Trans. Auto. Control AC-15, 3-10 (1970).

[12] R. S. Bucy and K. D. Senne, "Realization of Optimum Discrete-Time Nonlinear Estimators," Proc. Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, September 1970, 6-17.

[13] D. L. Alspach, "A Bayesian Approximation Technique for Estimation and Control of Time Discrete Stochastic Systems," Ph.D. Dissertation, Univ. of California, San Diego (1970).

[14] D. L. Alspach and H. W. Sorenson, "Approximation of Density Functions by a Sum of Gaussians for Nonlinear Bayesian Estimation," Proc. Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, September 1970, 19-31.

[15] R. S. Bucy, R. A. Geesey, and K. D. Senne, "Passive Receiver Design via Nonlinear Filtering Theory," Proc. Third Hawaii International Conf. on System Sciences, Vol 1, (1970) 477-480.

[16] H. W. Sorenson and D. L. Alspach, "Recursive Bayesian Estimation Using Gaussian Sums," Automatica 7, 465-479 (1971).

[17] E. Tse, "Parallel Computation of the Conditional Mean State Estimate for Nonlinear Systems," Proc. Second Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, September 1971, 385-394.

[18] R. S. Bucy and K. D. Senne, "Digital Synthesis of Nonlinear Filters," Automatica 7, 287-298 (1971).

[19] R. S. Bucy and K. D. Senne, "A Two-Dimensional Passive Ranging Experiment using Optimal Nonlinear Filtering," Air Force Weapons Laboratories Computer Films No. 71-0330-02, March 1971.

[20] R.J.P. deFigueiredo and Y. G. Jan, "Spline Filters," Proc. Second Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, September 1971, 88-89.

[21] C. Hecht, "Digital Realization of Non-Linear Filters," Proc. Second Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, Sept. 1971, 152-158.

[22] R. S. Bucy, M.J. Merritt, and D.S. Miller, "Hybrid Computer Synthesis of Optimal Discrete Nonlinear Filters," Proc. Second Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, Sept. 1971, 59-87.

[23] K. D. Senne, "Computer Experiments with Nonlinear Filters," Proc. Second Symp. on Nonlinear Estimation and Its Applications, San Diego, 1971, 314-324.

[24] D. S. Miller, "Hybrid Synthesis of Optimal Discrete Nonlinear Filters," Ph.D. Dissertation, Univ. of Southern California, 1971.

[25] J. L. Center, "Practical Nonlinear Filtering of Discrete Observations by Generalized Least Squares Approximation of the Conditional Probability Distribution," Proc. Second Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, Sept. 1971, 88-99.

[26] H. L. Weinert and T. Kailath, "Stochastic Interpolation and Recursive Algorithms for Spline Functions," Annals of Statistics, 2,4, 787-794 (1974).

[27] R. S. Bucy, "Realization of Non-Linear Filters," Proc. Second Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, Sept. 1971, 51-58.

[28] R. S. Bucy, "Building and Evaluating Non-Linear Filters," Proc. Symp. on Appl. Match.; Stochastic Diff. Eqns., Amer. Math. Soc., April 1972.

[29] K. D. Senne, "Bayes Law Implementation: Optimal Discrete-Time Phase Estimation," Proc. SWIEEECO Conf., Dallas, April 1972.

[30] C. Hecht, "Synthesis and Realization of Nonlinear Filters," Ph.D. Dissertation, Univ. of Southern California, 1972.

[31] R. S. Bucy, C. Hecht, and K. D. Senne, "Optimal Phase Demodulation via Discrete Nonlinear Filtering," Air Force Weapons Laboratory Computer Films No. 72-0401-01, Apr. 1972.

[32] R. S. Bucy, C. Hecht, and K. D. Senne, "An Engineer's Guide to Building Nonlinear Filters," Final Report SRL-TR-72-0004, Project 7904, Frank J. Seiler Research Laboratory, USAF Academy, Colorado, (1972), DDC-AD-746921/2.

[33] W. J. Bouknight, et al, "The Illiac-IV Systems," Proc. IEEE 60, 369-388 (1972).

[34] K. D. Senne, "A Machine Independent Monte Carlo Evaluation of the Performance of Dynamic Systems," Stochastics, 1, 3, 215-238, (1974).

[35] R. S. Bucy, C. Hecht, and K. D. Senne, "New Methods for Nonlinear Filtering," Rev. Francais d'Automatique, J-1, 3-54 (1973).

[36] R. S. Bucy and H. Youssef, "Fourier Realization of the Optimal Phase Demodulator," Proc. 4th Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, (Western Periodicals, 1973) 34-38.

[37] R. S. Bucy, M. J. Merritt, and D. S. Miller, "Hybrid Synthesis of the Optimal Discrete Filter," Stochastics, 1, 151-211 (1974).

[38] L. Basañez, P. Brunet, R. S. Bucy, R. Huber, D. S. Miller, and J. Pages, "Hybrid Computer Optimal Filter," Proc. 6th Symp. on Nonlinear Estimation Theory and Its Applications, San Diego, California, 1975.

[39] H. M. Youssef, "Suboptimal Phase Demodulation," Proc. 6th Symp. on Nonlinear Estimation and Its Applications, San Diego, California, 1975.

[40] H. Youssef, "Interpolative Spline Filters," Ph.D. Thesis, Aerospace Eng. Dept. Univ. of Southern California (1975).

[41] R. S. Bucy and H. Youssef, "Dependence of the Optimal Phase Demodulator on Statistical Parameters," IEEE Trans. Autom. Contr., AC-20, 2, 259-260 (1975).

[42] D. A. Calahan, W. N. Joy and D. A. Orbits, "Preliminary report on results of matrix benchmarks on vector processors," Report #94, System Engineering Laboratory, Univ. of Mich. (May 1976).

[43] L. Basañez, P. Brunet, R. S. Bucy, R. Huber, D. S. Miller, and J. Pages, "Simulation and Implementation of a Hybrid Computer Algorithm for an Optimal Nonlinear Filtering," Proc. 9th Symp. on System Science, Honolulu, Hawaii, 1976.

[44] R. S. Bucy and H. Youssef, "Optimal phase demodulation," IEEE Trans. Autom. Contr., AC-21, 5, 732-736 (1976).

[45] R. S. Bucy, K. D. Senne and H. Youssef, "Parallel, Pipeline and Serial Realization of Optimal Demodulators," Stochastic Control, Editors Roxin and Sternberg (Marcel Dekker, New York, 1977).

[46] System Guide for the Illiac IV User, Institute for Advanced Computation, Ames Research Center, Moffett Field, CA 94035.

[47] Processor Handbook, Software Development Packages, Floating Point Systems AP-120B, 7259-02, Beaverton, Oregon.

[48] CRAY-1 Computer Reference Manual 220004, Cray Research, Inc. Bloomington, Min. (1976).

[49] STAR Reference Manual, NASA Langley Research Center, Hampton, Va.

[50] D. Casseres, "Illiac IV Machine Reference Manual for the Programmer," Inst. for Advanced Comp. Doc. PG-11700-0000-A, (June 1975).

[51] STAR-100 Computer System Reference Manual, Control Data Corporation, Publication No. 602560000, 1975.

[52] STAR Programming Manual, NASA, Langley Research Center, Hampton, VA (March 5, 1976).

[53] P. M. Johnson, "An Introduction to Vector Processing," Computer Design, 89-97 (Feb. 1978).

[54] C. Jensen, "Taking Another Approach to Supercomputing," Datamation, 159-172 (Feb. 1978).

[55] W. R. Wittmayer, "Array Processor Provides High Throughput Rates," Computer Design, 93-100 (Mar. 1978).

[56] L. Higbie, "Applications of Vector Processing," Computer Design, 139-145 (April 1978).

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-78-98	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* New Frontiers in Nonlinear Filtering		5. TYPE OF REPORT & PERIOD COVERED Technical Note 6. PERFORMING ORG. REPORT NUMBER Technical Note 1978-16
7. AUTHOR*(s)* Richard S. Bucy and Kenneth D. Senne		8. CONTRACT OR GRANT NUMBER*(s)* F19628-78-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element No. 65705F Project No. 649L
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Systems Command, USAF Andrews AFB Washington, DC 20331		12. REPORT DATE 26 May 1978 13. NUMBER OF PAGES 60
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB Bedford, MA 01731		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

phase demodulation
Monte Carlo performance analysis
nonlinear filter
computer architectures
parallel computer architectures

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Examples of two and three dimensional phase demodulation problems are presented. Computer realizations for the optimal nonlinear phase estimator are discussed in detail, with emphasis on parallel computer architectures. Implementation of the nonlinear filter on various computer architectures, including the CDC6600/7600, CDC STAR-100, Illiac IV, the CRAY-1, and the Floating Point System AP120B is reviewed. Detailed Monte Carlo performance analysis is presented for the two-dimensional system, while partial results are included for the three dimensional case. Implications concerning the ideal computer architecture for nonlinear filter realization are discussed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*